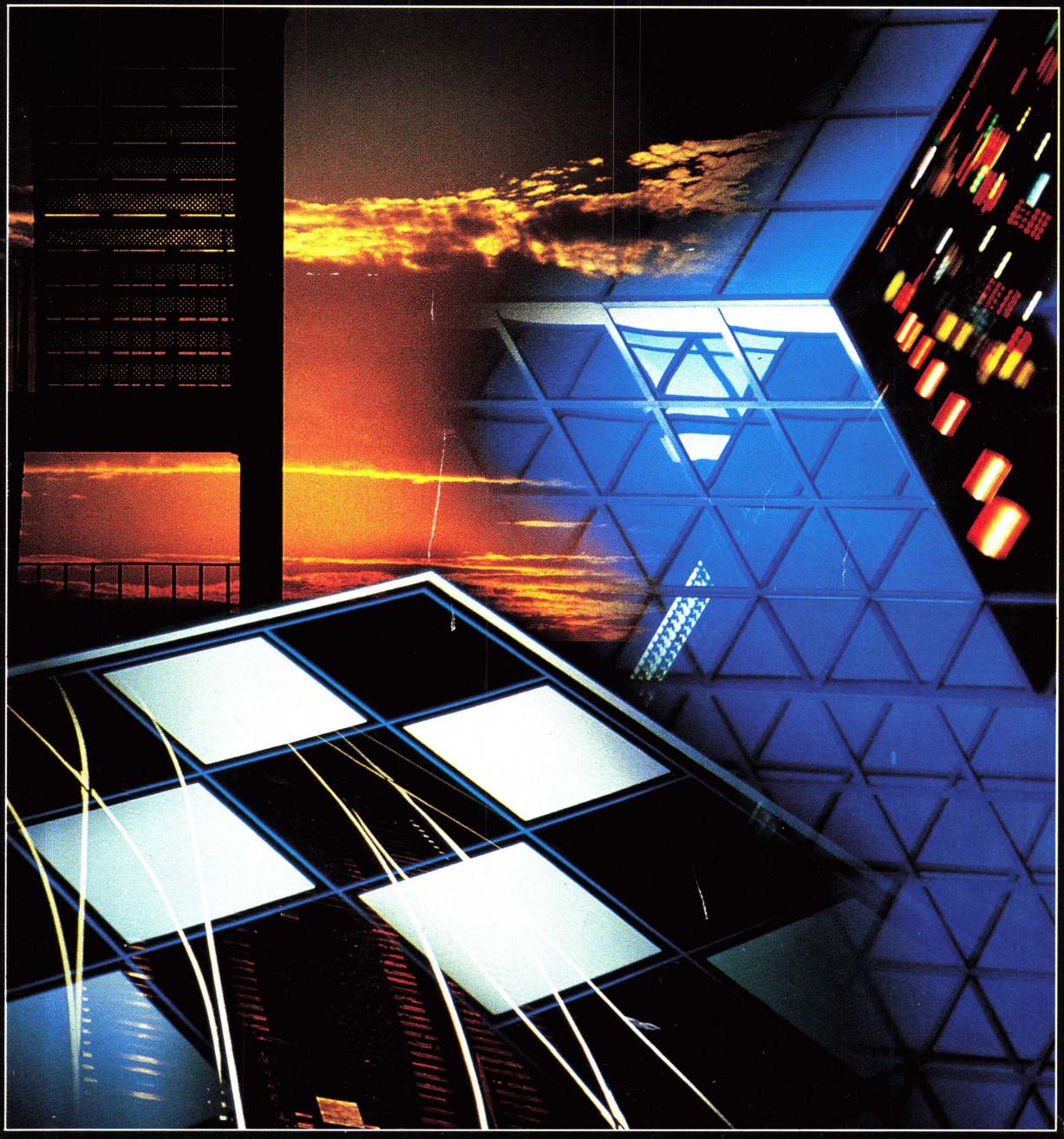
intel

Programmable Logic Handbook



Order Number: 296083-003

LITERATURE

To order Intel literature write or call:

Intel Literature Sales
P.O. Box 58130
Santa Clara, CA 95052-8130

Toll Free Number:
(800) 548-4725*

Use the order blank on the facing page or call our **Toll Free** Number listed above to order literature. Remember to add your local sales tax and a 10% postage charge for U.S. and Canada customers, 20% for customers outside the U.S. Prices are subject to change.

1988 HANDBOOKS

Product line handbooks contain data sheets, application notes, article reprints and other design information.

NAME	ORDER NUMBER	**PRICE IN U.S. DOLLARS
COMPLETE SET OF 8 HANDBOOKS Save $50.00 off the retail price of $175.00	**231003**	**$125.00**
AUTOMOTIVE HANDBOOK (Not included in handbook Set)	231792	**$20.00**
COMPONENTS QUALITY/RELIABILITY HANDBOOK (Available in July)	210997	**$20.00**
EMBEDDED CONTROLLER HANDBOOK (2 Volume Set)	210918	**$23.00**
MEMORY COMPONENTS HANDBOOK	210830	**$18.00**
MICROCOMMUNICATIONS HANDBOOK	231658	**$22.00**
MICROPROCESSOR AND PERIPHERAL HANDBOOK (2 Volume Set)	230843	**$25.00**
MILITARY HANDBOOK (Not included in handbook Set)	210461	**$18.00**
OEM BOARDS AND SYSTEMS HANDBOOK	280407	**$18.00**
PROGRAMMABLE LOGIC HANDBOOK	296083	**$18.00**
SYSTEMS QUALITY/RELIABILITY HANDBOOK	231762	**$20.00**
PRODUCT GUIDE Overview of Intel's complete product lines	210846	**N/C**
DEVELOPMENT TOOLS CATALOG	280199	**N/C**
INTEL PACKAGING OUTLINES AND DIMENSIONS Packaging types, number of leads, etc.	231369	**N/C**
LITERATURE PRICE LIST List of Intel Literature	210620	**N/C**

*Good in the U.S. and Canada

**These prices are for the U.S. and Canada only. In Europe and other international locations, please contact your local Intel Sales Office or Distributor for literature prices.

About Our Cover:

Programmable logic devices are a seeming contradiction: standard devices that offer custom solutions. Individual implementations are almost infinite in their variability, like the apparently simple game of chess.

intel

LITERATURE SALES ORDER FORM

NAME: _____

COMPANY: _____

ADDRESS: _____

CITY: _____ STATE: _____ ZIP: _____

COUNTRY: _____

PHONE NO.: ( ) _____

ORDER NO.	TITLE	QTY.	PRICE	TOTAL

Subtotal _____

Must Add Your Local Sales Tax _____

Must add appropriate postage to subtotal (10% U.S. and Canada, 20% all other) → Postage _____

Total _____

Pay by Visa, MasterCard, American Express, Check, Money Order, or company purchase order payable to Intel Literature Sales. Allow 2-4 weeks for delivery.

☐ Visa ☐ MasterCard ☐ American Express Expiration Date _____

Account No. _____

Signature: _____

Mail To: Intel Literature Sales
P.O. Box 58130
Santa Clara, CA
95052-8130

International Customers outside the U.S. and Canada should contact their local Intel Sales Office or Distributor listed in the back of most Intel literature.

European Literature Order Form in back of book.

Call Toll Free: **(800) 548-4725** for phone orders

Prices good until 12/31/88.

Source HB

Intel Literature **update** Service

Get Intel's Latest Technical Literature, Automatically!

Exclusive, Intel Literature Update Service

Take advantage of Intel's year-long, low cost Literature Update Service and you will receive your first package of information followed by automatic quarterly updates on all the latest product and service news from Intel.

Choose one or all five product categories update

Each product category update listed below covers in depth, all the latest Handbooks, Data Sheets, Application Notes, Reliability Reports, Errata Reports, Article Reprints, Promotional Offers, Brochures, Benchmark Reports, Technical Papers and much more . . .

1. Microprocessors

Product line handbooks on Microprocessors, Embedded Controllers and Component Quality/Reliability, *Plus,* the Product Guide, Literature Guide, Packaging Information and 3 quarterly updates. $70.00 Order Number: **555110**

2. Peripherals

Product line handbooks on Peripherals, Microcommunications, Embedded Controllers, and Component Quality/Reliability, *Plus,* the Product Guide, Literature Guide, Packaging Information and 3 quarterly updates. $50.00 Order Number: **555111**

3. Memories

Product line handbooks on Memory Components, Programmable Logic and Components Quality/Reliability, *Plus,* the Product Guide, Literature Guide, Packaging Information and 3 quarterly updates. $50.00 Order Number: **555112**

4. OEM Boards and Systems

Product line handbooks on OEM Boards & Systems, Systems Quality/Reliability, *Plus,* the Product Guide, Literature Guide, Packaging Information and 3 quarterly updates. $50.00 Order Number: **555113**

5. Software

Product line handbooks on Systems Quality/Reliability, Development Tools Catalog, *Plus,* the Product Guide, Literature Guide, Packaging Information and 3 quarterly updates. $35.00 Order Number: **555114**

To subscribe, rush the Literature Order Form in this handbook, or call today, toll free (800) 548-4725.*

Subscribe by March 31, 1988 and receive a valuable free gift.

The charge for this service covers our printing, postage and handling cost only.

Please note: Product manuals are not included in this offer.

Customers outside the U.S. and Canada should order directly from the U.S.

Offer expires 12/31/88.

*Good in the U.S. and Canada.

intel

CUSTOMER SUPPORT

CUSTOMER SUPPORT

Customer Support is Intel's complete support service that provides Intel customers with hardware support, software support, customer training, and consulting services. For more information contact your local sales offices.

After a customer purchases any system hardware or software product, service and support become major factors in determining whether that product will continue to meet a customer's expectations. Such support requires an international support organization and a breadth of programs to meet a variety of customer needs. As you might expect, Intel's customer support is quite extensive. It includes factory repair services and worldwide field service offices providing hardware repair services, software support services, customer training classes, and consulting services.

HARDWARE SUPPORT SERVICES

Intel is committed to providing an international service support package through a wide variety of service offerings available from Intel Hardware Support.

SOFTWARE SUPPORT SERVICES

Intel's software support consists of two levels of contracts. Standard support includes TIPS (Technical Information Phone Service), updates and subscription service (product-specific troubleshooting guides and COMMENTS Magazine). Basic support includes updates and the subscription service. Contracts are sold in environments which represent product groupings (i.e., iRMX environment).

CONSULTING SERVICES

Intel provides field systems engineering services for any phase of your development or support effort. You can use our systems engineers in a variety of ways ranging from assistance in using a new product, developing an application, personalizing training, and customizing or tailoring an Intel product to providing technical and management consulting. Systems Engineers are well versed in technical areas such as microcommunications, real-time applications, embedded microcontrollers, and network services. You know your application needs; we know our products. Working together we can help you get a successful product to market in the least possible time.

CUSTOMER TRAINING

Intel offers a wide range of instructional programs covering various aspects of system design and implementation. In just three to ten days a limited number of individuals learn more in a single workshop than in weeks of self-study. For optimum convenience, workshops are scheduled regularly at Training Centers worldwide or we can take our workshops to you for on-site instruction. Covering a wide variety of topics, Intel's major course categories include: architecture and assembly language, programming and operating systems, bitbus and LAN applications.

Intel the Microcomputer Company:

When Intel invented the microprocessor in 1971, it created the era of microcomputers. Whether used as microcontrollers in automobiles or microwave ovens, or as personal computers or supercomputers, Intel's microcomputers have always offered leading-edge technology. In the second half of the 1980s, Intel architectures have held at least a 75% market share of microprocessors at 16 bits and above. Intel continues to strive for the highest standards in memory, microcomputer components, modules, and systems to give its customers the best possible competitive advantages.

PROGRAMMABLE LOGIC HANDBOOK

1988

intel

Intel Corporation makes no warranty for the use of its products and assumes no responsibility for any errors which may appear in this document nor does it make a commitment to update the information contained herein.

Intel retains the right to make changes to these specifications at any time, without notice.

Contact your local sales office to obtain the latest specifications before placing your order.

The following are trademarks of Intel Corporation and may only be used to identify Intel Products:

Above, BITBUS, COMMputer, CREDIT, Data Pipeline, FASTPATH, GENIUS, i, $\hat{i}$, ICE, iCEL, iCS, iDBP, iDIS, I^2ICE, iLBX, i_m, iMDDX, iMMX, Inboard, Insite, Intel, int$_e$l, int$_e$lBOS, Intel Certified, Intelevision, int$_e$ligent Identifier, int$_e$ligent Programming, Intellec, Intellink, iOSP, iPDS, iPSC, iRMK, iRMX, iSBC, iSBX, iSDM, iSXM, KEPROM, Library Manager, MAP-NET, MCS, Megachassis, MICROMAINFRAME, MULTIBUS, MULTICHANNEL, MULTIMODULE, MultiSERVER, ONCE, OpenNET, OTP, PC-BUBBLE, Plug-A-Bubble, PROMPT, Promware, QUEST, QueX, Quick-Pulse Programming, Ripplemode, RMX/80, RUPI, Seamless, SLD, SugarCube, SupportNET, UPI, and VLSiCEL, and the combination of ICE, iCS, iRMX, iSBC, iSBX, iSXM, MCS, or UPI and a numerical suffix, 4-SITE.

MDS is an ordering code only and is not used as a product name or trademark. MDS® is a registered trademark of Mohawk Data Sciences Corporation.

*MULTIBUS is a patented Intel bus.

Additional copies of this manual or other Intel literature may be obtained from:

Intel Corporation
Literature Distribution
Mail Stop SC6-59
3065 Bowers Avenue
Santa Clara, CA 95051

© INTEL CORPORATION 1987

Table of Contents

Alphanumeric Index vii

CHAPTER 1

Overview

Overview 1-1

CHAPTER 2

EPLDs—Erasable Programmable Logic Devices

DATA SHEETS

Data Sheet Specifications 2-1

5C031, 300-Gate CHMOS H-Series Erasable Programmable Logic Device (H-EPLD) 2-2

5C032, 300-Gate CHMOS H-Series Erasable Programmable Logic Device (H-EPLD) 2-14

5C060/5C090, 600-/900-Gate CHMOS H-Series Erasable Programmable Logic Device (H-EPLD) 2-27

5C121, 1200-Gate CHMOS H-Series Erasable Programmable Logic Device 2-45

5C180, 1800-Gate CHMOS Erasable Programmable Logic Device 2-61

5AC312, Erasable Programmable Logic Device 2-91

APPLICATION BRIEFS

AB-8 Implementing Cascaded Logic in the 5C121 2-108

AB-9 5C121 As a Three and One-Half Digit Display Driver 2-113

AB-10 Square Pegs in Round Holes—A Fitting Tutorial for the 5C121 2-118

AB-11 16-Bit Binary Counter Implementation Using the 5C060 EPLD 2-130

AB-12 Designing a Mailbox Memory for Two 5C031s 2-140

AB-16 Atypical Latch/Register Construction in EPLDs 2-154

AB-18 TTL Macro Library Listing for EPLD Designs 2-161

APPLICATION NOTES

AP-271 Applying the 5C121 Architecture 2-165

AP-272 The 5C060 Unification of a CHMOS System 2-177

AP-276 Implementing a CMOS Bus Arbiter/Controller in the 5C060 EPLD 2-188

AP-304 Simulation of EPLD Timing 2-198

AP-307 EPLDs, PLAs, and TTL—Comparing the "Hidden Costs" in Production 2-212

TECHNICAL PAPERS

Techniques for Modular EPLD Designs 2-234

ARTICLE REPRINTS

AR-450 Crosspoint Switch: A PLD Approach 2-244

AR-451 A Programmable Logic Mailbox for 80C31 Microcontrollers 2-248

AR-454 Regain Lost I/O Ports with Erasable PLDs 2-251

CHAPTER 3

Advanced Architecture EPLDs

DATA SHEETS

5CBIC, Programmable BUS Interface Controller 3-1

APPLICATION NOTES

AP-305 Dual-Port Memory Control Using the 5CBIC 3-19

AP-308 The Multiplexed BUS Interface with the 5CBIC 3-26

AP-309 DRAM Address Interface with the 5CBIC 3-34

ARTICLE REPRINTS

AR-453 Programmable Logic Shrink Bus Interface Designs 3-39

CHAPTER 4

Development Support Tools

DATA SHEETS

iPLDS II, The Intel Programmable Logic Development System Version II 4-1

Table of Contents (Continued)

iUP-PC, Intel Universal Programmer for the Personal Computer 4-12
PRODUCT BRIEFS
SCHEMA II-PLD 4-18
Macro Librarian 4-19
UTILITIES
Functional Simulator Utility 4-20
PAL2ADF Utility 4-21
JED2HEX Conversion Utility 4-24
APPLICATION NOTES
AP-279 Implementing an EPLD Design Using Intel's Programmable Logic Development System 4-25
AP-311 Using Macros in EPLD Designs 4-79
AP-312 Creating Macros for EPLD Designs 4-91
TECHNICAL PAPERS
Tools for Optimizing PLD Designs 4-101

CHAPTER 5
Appendix
Second Source Cross Reference 5-1
PLA to EPLD Replacement 5-2
Ordering Information 5-3
Device Feature Comparison 5-4
EPLD Customer Support 5-5
Compatible Computers for iPLDS II 5-6

Alphanumeric Index

5AC312, Erasable Programmable Logic Device 2-91
5CBIC, Programmable BUS Interface Controller 3-1
5C031, 300-Gate CHMOS H-Series Erasable Programmable Logic Device (H-EPLD) 2-2
5C032, 300-Gate CHMOS H-Series Erasable Programmable Logic Device (H-EPLD) 2-14
5C060/5C090, 600-/900-Gate CHMOS H-Series Erasable Programmable Logic Device (H-EPLD) 2-27
5C121, 1200-Gate CHMOS H-Series Erasable Programmable Logic Device 2-45
5C180, 1800-Gate CHMOS Erasable Programmable Logic Device 2-61
iPLDS II, The Intel Programmable Logic Development System Version II 4-1
iUP-PC, Intel Universal Programmer for the Personal Computer 4-12

Overview

1

INTRODUCTION

In today's increasingly competitive marketplace, system designers need to squeeze out every little edge they can get from their designs. This has led to a trend towards better performance, smaller system sizes, lower power requirements and greater system reliability with a strong emphasis on preventing easy duplication of the system design. This trend provided the impetus to the system designers to move away from standard SSI and MSI logic components (54/74 & 4000 series Bipolar and CMOS families) towards a growing class of IC devices variously called 'ASIC' (application specific IC), 'USIC' (user specific IC) or, as referred to in this document, user defined logic.

User defined logic circuits allow system designers, for the first time, to tailor the actual silicon building blocks used in their systems to their individual system needs and requirements. Such customization provides the needed performance, reliability and compactness as well as design security. Cost per gate of logic implemented is also greatly reduced when user defined logic solutions are chosen over standard components.

User defined logic has therefore emerged as the fastest growing segment of the semiconductor industry and has presented its users, the system designers, with a wide range of implementation alternatives namely, programmable logic, gate arrays, standard cell and full custom design. The tradeoffs between these alternatives involves time-to-market, one-time engineering charges, expected unit volume, ease of use of design tools and familiarity with the design methodology.

This document discusses the reasons for the trend to user defined logic devices, briefly describes some of the user defined logic implementation alternatives and covers details on programmable logic devices, the only alternative that is completely user implementable. Tools used to design with programmable logic are also discussed here.

Details on Intel's programmable logic product line, including device terminology and nomenclature, architectural features and development tool features are also described in this document.

WHY USER DEFINED LOGIC?

System designers prefer user customized ICs for the following reasons:

a. SMALLER SYSTEM SIZES: Customized components allow for reducing chip count and saving board space, resulting in smaller system physical dimensions.

b. LOWER SYSTEM COSTS: When custom LSI or VLSI components are used instead of standard SSI and MSI logic elements, there is a considerable saving in component cost per system, assembly and manufacturing cost, printed circuit board area and board costs and inventory costs.

c. HIGHER PERFORMANCE: Reduced number of ICs contributes to faster system speeds as well as lower power consumption.

d. HIGHER RELIABILITY: Since probability of failure is directly related to the number of ICs in the system, a system composed of customized LSI & VLSI chips is statistically much more reliable than the identical system made up of SSI/MSI devices.

e. DESIGN SECURITY: Systems designed with standard components can be replicated relatively easily whereas systems that contain user customized ICs cannot be copied because "reverse engineering" of the customized components is extremely difficult. Thus, use of customized ICs allows for the protection of proprietary designs.

f. INCREASED FLEXIBILITY: Customized components allow for the tailoring of systems to the end user's specific needs relatively easily. This also allows for upgradability and obsolescence protection.

USER DEFINED IC— IMPLEMENTATION ALTERNATIVES

Currently, the choices available to the system designer for customization of ICs (see Figure 1) are as follows:

(1) user programmable ICs—programmable logic devices

(2) mask programmable ICs—gate arrays

(3) standard cell based ICs

(4) full custom ICs

Alternatives (1) & (2) are usually called 'Semicustom' because in these methods only a few (less than three) of the mask layers involved in the manufacture of the IC, are customized to the users' specifications. The later two alternatives (3) & (4), involve customization of all mask layers required to manufacture the ICs to the users' specifications and are therefore called 'Custom'.

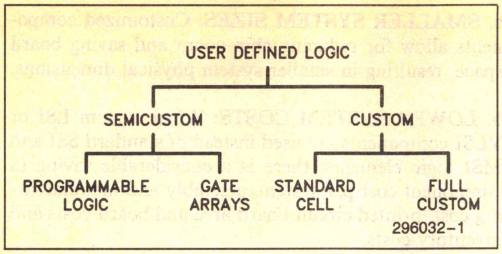


Figure 1. User Defined Logic Implementation Choices

PROGRAMMABLE LOGIC

Most user Programmable Logic Devices (PLD) are internally structured as variations of the PLA (programmable logic array) architecture, that is composed of an array of 'AND' gates connected to an array of 'OR' gates (see Figure 2). Programmable logic devices make use of the fact that any logic equation can be converted to an equivalent 'Sum-of-Products' form and can thus be implemented in the 'AND' and 'OR' architecture. This basic PLA structure has been augmented in most PLDs with input and output blocks containing registers, latches and feedback options, that let the user implement sequential logic functions in addition to combinational logic.

The number and locations of the programmable connections between the 'AND' and 'OR' matrices as well as the input and output blocks are predetermined by the architecture of the PLD. The user, depending on his logic requirements, determines which of these connections he would like to remain open and which he would like to close, through the programming of the PLD. Programmability of these connections is achieved using various memory technologies such as fuses, EPROM cells, EEPROM cells or Static RAM cells (see Figure 3).

User programmability allows for instant customization, very similar to user programmable memories such as PROMs or EPROMs. The user can purchase a PLD off-the-shelf, use a development system running on a personal computer and, in a matter of a few hours, have customized silicon in his hands. Figure 4 compares user-defined logic alternatives.

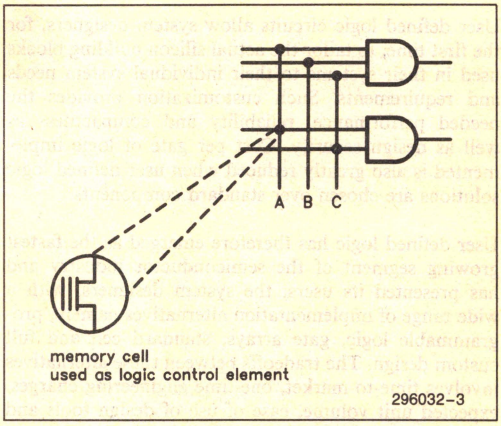


Figure 3. Programmable Connections

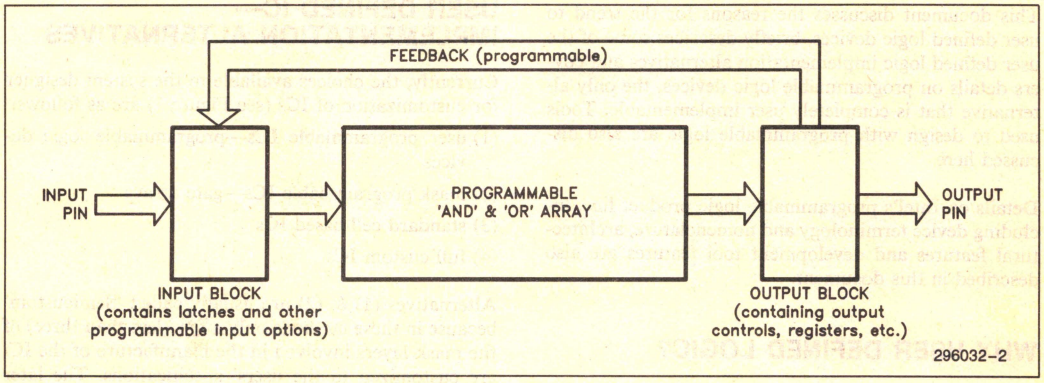


Figure 2. General Architecture of a PLD

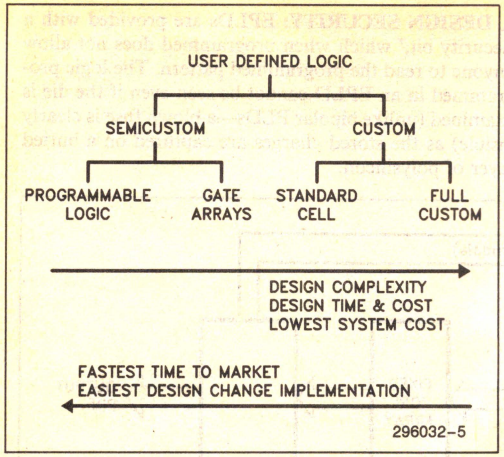


Figure 4. User Defined Logic Alternatives Compared

LIMITATIONS OF BIPOLAR FUSE TECHNOLOGY FOR PROGRAMMABLE LOGIC DEVICES

Until 1985, all PLDs were built using Bipolar fuse technology. The bipolar fuse based devices, although offering the users the benefits of quick time to market and low development costs, had several inherent limitations.

a. HIGH POWER CONSUMPTION: Bipolar processes by nature are power hungry and as a consequence also make for very hot systems, often requiring cooling aids such as heat sinks and fans. They also cannot operate at lower voltages (2–3V) and have a lower level of noise immunity than MOS devices.

b. LOWER INTEGRATION: A fuse takes up a large amount of silicon area; this fact in conjunction with the large power requirements makes for smaller levels of integration.

c. ONE-TIME PROGRAMMABILITY: Bipolar fuses can only be blown once and cannot be reprogrammed. This does not allow for easy prototyping and could result in significant losses when preprogrammed parts are inventoried and design changes occur.

d. TESTABILITY: Since fuses can only be blown once, bipolar PLDs can only be destructively tested. Thus, testing is usually done by sampling or through additional testing elements incorporated in the chips, which can be blown to examine electrical characteristics. However, such testing methods never allow for 100% testability of all parts shipped. Thus, most users of bipolar programmable logic devices resort to extensive post-programming testing, specific to their applications.

ERASABLE PROGRAMMABLE LOGIC DEVICES

Erasable programmable logic devices (EPLD) result from the matching of CHMOS EPROM technology with the architectures of programmable logic devices. EPLDs use EPROM cells as logic control elements and therefore, when housed in windowed ceramic packages, can be erased with UV light and reprogrammed. Figure 5 shows the architecture of Intel EPLDs.

Other than the obvious benefit of reprogrammability, EPLDs offer several very significant benefits over bipolar PLDs. These are:

1. LOW POWER CONSUMPTION: Due to the CMOS technology, these products consume an order of magnitude less power than the equivalent bipolar devices. This allows for the design of complete CMOS systems, that can operate at lower voltages (less than 5V). Also, this makes for cooler systems that do not require cooling systems like fans.

2. GREATER LOGIC DENSITY: EPROM cells are an order of magnitude smaller than the smallest fuses. This means that the same function can be accommodated in significantly smaller die area, or that greater amounts of logic can now be incorporated on a single chip. Thus higher integration programmable logic devices result with the use of EPROM elements.

3. TESTABILITY: Since the EPROM cells are erasable, the entire EPROM array of the EPLD can be 100% factory tested. Thus, before the part is shipped to the customers, it can be completely tested by the programming and erasure of all the EPROM logic control bits. This testing is therefore independent of any application, in contrast to the bipolar PLDs that need application specific testing.

4. ARCHITECTURAL ENHANCEMENTS: The inherent testability of the EPROM elements allows for

significant architectural improvements over bipolar PLDs. New features, such as buried registers, programmable registers, programmable clock control, etc., can now be incorporated because of this testability. These new features allow for greatly increased utilization of the EPLDs and use of these devices in newer applications.

5. DESIGN SECURITY: EPLDs are provided with a 'security bit,' which when programmed does not allow anyone to read the programmed pattern. The logic programmed in an EPLD cannot be seen even if the die is examined (unlike bipolar PLDs—a blown fuse is clearly visible) as the stored charges are captured on a buried layer of polysilicon.

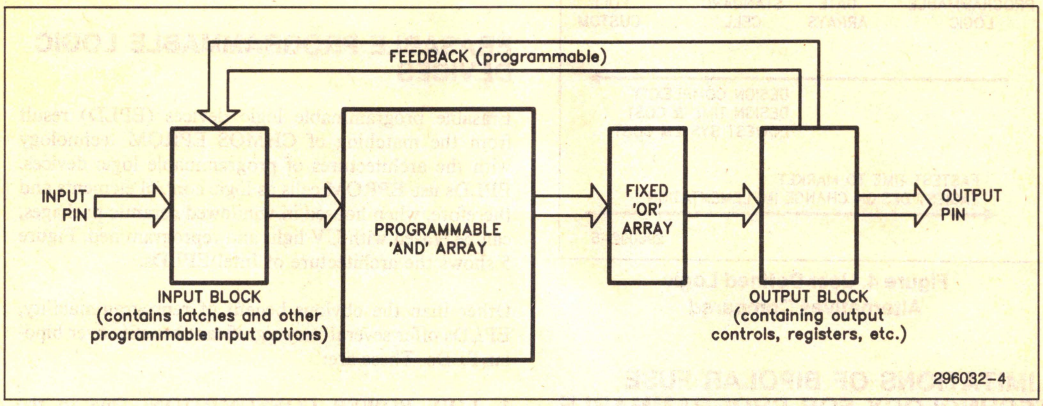


Figure 5. Architecture of Intel EPLDs

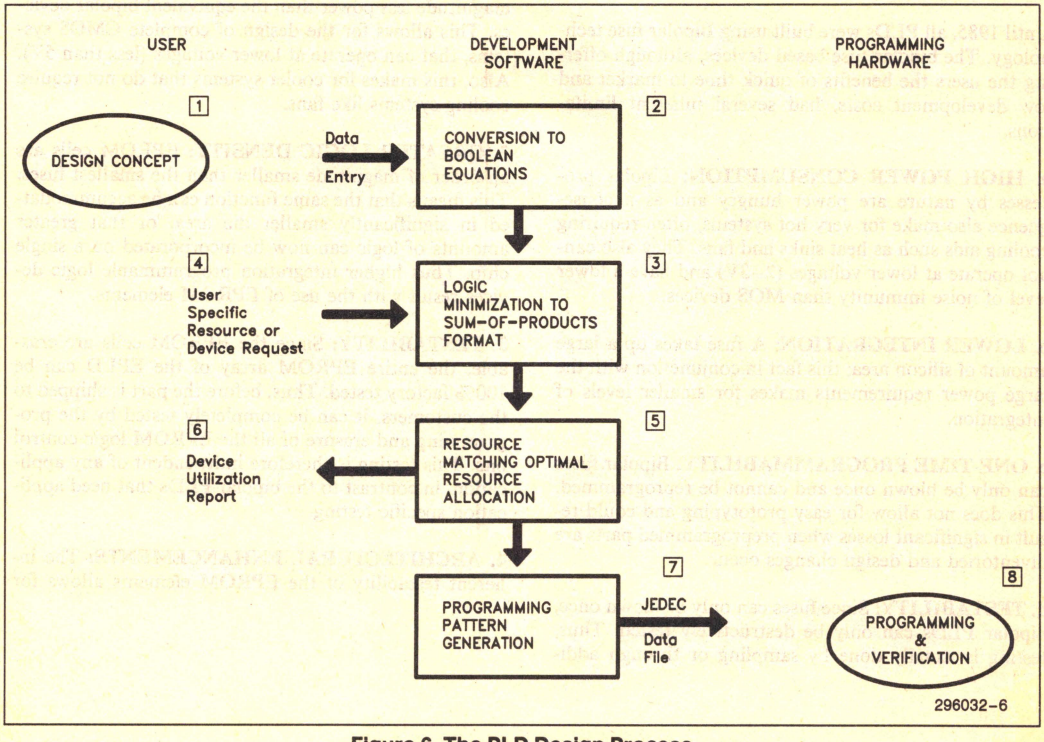


Figure 6. The PLD Design Process

The steps in a generalized design process of programmable logic is shown in Figure 6 and described in the following paragraphs.

STEP 1: The user decides on the logic he wants implemented in the PLD and enters the design into the PC or workstation. This **Design Entry** may be done by the following methods: (i)**SCHEMATIC CAPTURE**—A 'Mouse' or some other graphics input device is used to input schematics of the logic, (ii)**NET LIST ENTRY**—If the user has a hand drawn schematic he can enter the design into the computer by describing the symbols and interconnections in words using a standardized format called a net list (without using a graphics input device), (iii)**STATE EQUATION/DIAGRAM ENTRY**—Entry of a sequential design involving states and transitions between states. In the state diagram method circles represent states and the arrows interconnecting them represent the transitions. Equations or a state table can also be used to define a state machine, and (iv)**BOOLEAN EQUATIONS**—this is the most common design entry method. The logic is described in boolean algebraic equations.

STEP 2: The software converts all design entry data into boolean equations.

STEP 3: The boolean equations entered are converted to the sum of products format after logic reduction (minimization of the logic through heuristic algorithms).

STEP 4: The user has the ability to choose the PLD he would like the design implemented on. He can enter device choice and/or he can also enter in specific choices on the device as regards pinout he would like etc . . .

STEP 5: The software optimizes the logic equations to fit into the device using the minimum amount of resources (resources are input pins, output pins, registers and product terms and macrocells). This step is where the user requirements as regards required pins are taken into account. The user requests are viewed as constraints during the optimization process.

STEP 6: The software, at the end of the resource optimization/allocation, produces a report detailing the resources used up in fitting the design on the PLD. This report allows the user to incrementally stuff in logic by going back to Step 1 from this stage. Also, if the design overflowed the PLD, i.e., did not fit in the user chosen device, the software lists out the resources needed to complete the fit. The requirements such as four more inputs, one register more and one more output (are needed to complete the design) gives the user data in choosing a bigger PLD or in partitioning the intial design to fit into two devices.

STEP 7: The next step is to generate the appropriate programming pattern for the PLD. This is a standard "JEDEC" format interface and allows the output of the design software to be compatible with any piece of PROM programming hardware.

STEP 8: PROM programmer is used to program the pattern stored in the JEDEC file onto the PLD. Also, at this stage fuse programmed PLDs (bipolar) are functionally tested using test vectors included in the JEDEC file information.

CHMOS TECHNOLOGY IN EPLDs

EPLDs are manufactured with Intel's proprietary CHMOS (Complementary High Performance MOS) technology. The backbone of the process is the integration of both a P and an N channel MOS transistor on the same substrate. In addition, EPLD's programmable architecture makes use of Intel's proven EPROM cell for programmable array interconnections as well as macrocell configuration bits. These cells are programmed electrically and erased with ultraviolet light. For details on Intel's CHMOS technology and EPROM cells technology, refer to the *Components Quality/Reliability Handbook*, Order Number 210997.

CHMOS DESIGN GUIDELINES

Designing with Intel EPLDs is relatively straightforward if the following guidelines are observed:

- Minimize the occurrence of ESD (electro-static discharge) when storing or handling EPLDs.
- Observe good design rules in printed circuit board layout.
- Provide adequate decoupling capacitance at both the device and the board level.
- Connect all unused inputs to V_{CC} or GND (CHMOS inputs should not be left floating).

Electrostatic Discharge

The two most common sources of electrostatic discharge are the human body and a charged environment.

A charged human body that touches a device lead discharges electricity into the device. Electrostatic discharge from people handling devices has long been recognized by manufacturers and users of all MOS products. Human body static electricity can be controlled by using ground straps and anti-static spray on carpeted floors. CHMOS devices should also be stored and carried in conductive tubes or anti-static foam to minimize exposure to ESD from people.

Discharge also occurs when an integrated circuit is charged to one potential and then contacts a conductor at another potential. This type of ESD can be reduced

by grounding all work surfaces, grounding all handling equipment, removing static generators such as paper from the work area, and erasing EPLDs in metal tubes, metal trays, or conductive foam.

PCB Layout

The best PCB performance is obtained when close attention is payed to V_{CC}, GND, and signal traces. V_{CC} and GND should be gridded to minimize inductive reactance and to approximate a trace layer. Clocks should be layed out to minimize crosstalk. Ensure adequate power supply and ground pins on the board connector.

Decoupling

Decouple each EPLD with a ceramic capacitor in the range of 0.01 to 0.2 μF, depending on board frequency and current consumption. For most applications, a 0.1 μF capacitor will suffice. The following equation produces the exact value:

$$C = \frac{\Delta I_{CC}}{\Delta V / \Delta T}$$

where C = capacitor value

ΔI_{CC} = maximum switched current

ΔV = switching level

ΔT = switching time

For boards that contain mixed logic (EPLDs and TTL), observe both EPLD and TTL decoupling practices.

Unused Inputs

To minimize noise receptivity and power consumption, all unused inputs to EPLDs should be connected to V_{CC} or GND. By default, iPLS II software assigns unused inputs to GND. These pins, shown on the pinout representation of the iPLS II report file, should be connected to ground on the PCB. Pins listed as RESERVED on the report file must be left floating. Pins marked N.C. have no internal device connections and can also be left floating.

BOOLEAN MINIMIZATION TECHNIQUES FOR PLA ARCHITECTURES

Minimization plays an important role in logic design. Methods for minimization can be grouped into two classes. Class 1 includes manual methods for minimization, such as Boolean reduction or Karnaugh mapping. Class 2 is computer-assisted minimization.

Tabular methods like Karnaugh maps are efficient up to a certain point. Past that point, however, computer-assisted minimization plays a crucial part in efficient design. Even at the computer-assisted stage, the choice of minimizer software has an impact on time and the confidence level of the reduced equation (i.e., is it in the smallest possible form).

iPLS II software includes a minimizer that uses the ESPRESSO algorithms. ESPRESSO was developed by U.C. Berkeley during the summers of 1981 and 1982 in an effort to study the various strategies used by the MINI logic minimizer developed by IBM , [HON 74] and PRESTO developed by D. Brown [BRO 81]. ESPRESSO uses many of the core principles in MINI and PRESTO while improving on the speed and efficiency of their algorithms.

The primary advantage of the ESPRESSO minimizer becomes apparent when designing large finite state machines or complex, product-term intensive logic designs. In these cases, ESPRESSO arrives at the minimize solution sooner, and frequently reduces the logic to a smaller number of product terms. In certain cases where other CAD packages such as ABEL™ (PRESTO) or CUPL™ minimize equations to greater than 8 product terms, iPLS II further reduces these equations to allow the design to fit into devices supporting up to 8 product terms.

For more information on ESPRESSO, refer to *Logic Minimization Algorithms for VLSI Synthesis*, Brayton, Hachtel, McMullen, and Sangiovanni-Vincentelli, Kluwer Academic Publishers.

References

[BRO 81] D.W. Brown, "A State-Machine Synthesizer—SMS", Proc. 18th Design Automation Conference, pp. 301–304. Nashville, June 1981.

[HON 74] S. J. Hong, R. G. Cain and D. L. Ostapko, "MINI: A heuristic approach to logic minimization." *IBM Journal of Research and Development*, Vol. 18, pp. 443–458, September 1974.

ABEL™ is a trademark of Data I/O Corporation

CUPL™ is a trademark of Personal CAD Systems, Inc.

LOGIC REFRESHER COURSE

Minimization of EPLD logic equations is normally performed by sophisticated algorithms that eliminate the need for tedious manual reductions. The sections provided here contain logic reference tables for cases where manual reduction techniques may be desirable.

Boolean Algebra

The Sum-of-Product architecture used in EPLDs makes Boolean algebra ideal for design analysis. The following tables summarize standard Boolean functions.

Properties

$A * B = B * A$		Commutative Property
$A + B = B + A$		
$A * (B * C) = (A * B) * C$		Associative Property
$A + (B + C) = (A + B) + C$		
$A * (B + C) = A * B + A * C$		Distributive Property
$A + B * C = (A + B) * (A + C)$		

Postulates

$0 * 0 = 0$	$0 + 0 = 0$	$\overline{0} = 1$
$0 * 1 = 0$	$0 + 1 = 0$	$\overline{1} = 0$
$1 * 1 = 1$	$1 + 1 = 1$	

Theorems

$A * 0 = 0$	$A + 0 = A$	$\overline{\overline{A}} = A$
$A * 1 = A$	$A + 1 = 1$	
$A * A = A$	$A + A = A$	
$A * \overline{A} = 0$	$A + \overline{A} = 1$	

DeMorgan's Theorems

$(\overline{A + B + C + D})$	=	$\overline{A} * \overline{B} * \overline{C} * \overline{D}$
$(\overline{A * B * C * D})$	=	$\overline{A} + \overline{B} + \overline{C} + \overline{D}$

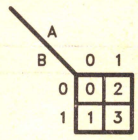
Logic Functions

$A * A$	=	A **AND** A
$A + A$	=	A **OR** A
$\overline{A}$	=	A **NOT**
$A \oplus B$ = A **EXCLUSIVE OR** B	=	$A\overline{B} + \overline{A}B$

Karnaugh Maps

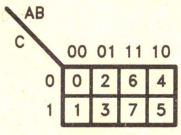
Graphical representation of data is usually easier to analyze than strings of ones and zeros. The Karnaugh Map techniques take advantage of this capability and provide an important tool to the logic designer.

Two Variables



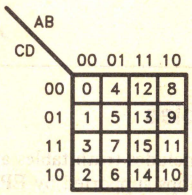
296032-7

Three Variables



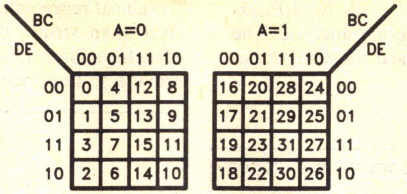
296032-8

Four Variables



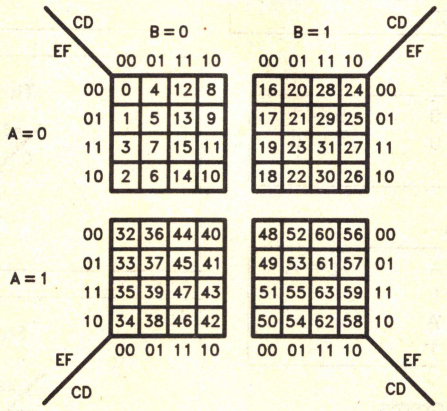
296032-9

Five Variables



296032-10

Six Variables



296032-11

Flip-Flop Tables

This subsection includes truth tables and excitation tables for the flip-flops supported by EPLDs.

D Truth Table

D	Q_N	Q_{N+1}
0	0	0
0	1	0
1	0	1
1	1	1

D Excitation Table

Q_N	Q_{N+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

T Truth Table

T	Q_N	Q_{N+1}
0	0	0
0	1	1
1	0	1
1	1	0

T Excitation Table

Q_N	Q_{N+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

JK Truth Table

J	K	Q_N	Q_{N+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

JK Excitation Table

Q_N	Q_{N+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

SR Truth Table

S	R	Q_N	Q_{N+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	Illegal	

JK Excitation Table

Q_N	Q_{N+1}	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

NOTES:
Q_N = Present State
Q_{N+1} = Next State
X = Don't Care

AUTOMATIC STANDBY MODE (TURBO BIT)

INTEL EPLDs contain a programmable bit, the Turbo Bit, that optimizes devices for speed or power savings. When TURBO = ON, EPLDs are optimized for speed. When TURBO = OFF, they are optimized for power savings by automatically entering standby mode when input transitions are not detected over a short period of time. The following paragraphs describe how the Turbo Bit affects power and speed in EPLDs.

Turbo Off (Low Power)

Intel EPLDs contain circuitry that monitors all inputs for transitions. When a transition is detected while the device is in standby mode, the circuit generates an active pulse. The leading edge of this pulse wakes the device up and the device responds according to its programming, changing outputs as necessary. If no new transitions occur during the active pulse, the device enters standby mode again. Outputs are always held valid in standby mode. Input transitions that occur during the active mode interval retrigger the active pulse. The active pulse is different depending on the device (5C060, 5AC312, etc), but is typically 2–4 times the propagation delay for a particular device.

In applications with infrequent input transitions, standby mode can result in significant power savings (see the appropriate data sheet for standby power vs. active power). The slight speed loss associated with waking up a device is in the range of 0–10 ns, which is small enough to allow standby mode to be used with most applications (see the appropriate data sheet for effect of Turbo Bit on performance).

Turbo On (Faster Speed)

In cases where the slight speed loss associated with waking a device from standby mode cannot be traded off to save power, the Turbo bit can be enabled for maximum speed operation. With the Turbo Bit enabled, the device is always in active mode, thus avoiding the wakeup delay. Note that data sheet performance is specified with the Turbo Bit enabled.

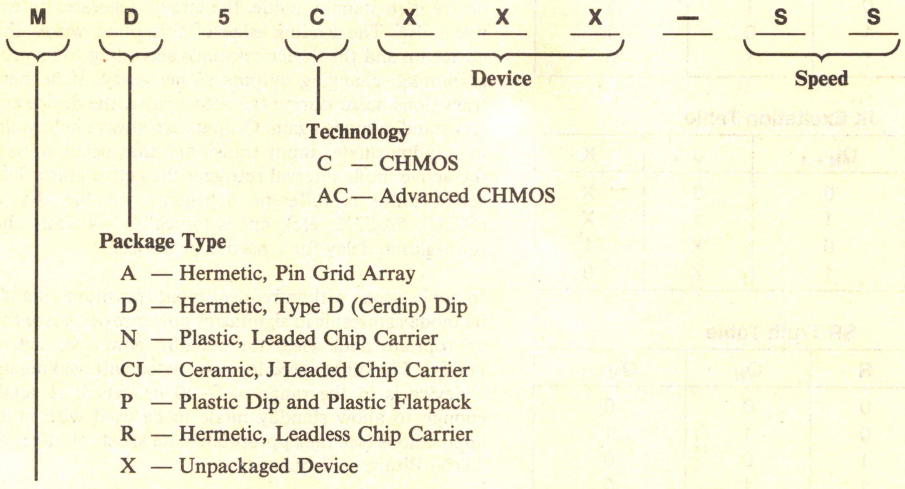
The Turbo Bit is enabled/disabled via a TURBO = ON or TURBO = OFF statement in an iPLS II ADF OPTIONS: statement. It can also be enabled/disabled by editing the JEDEC file using device programmable software. With TURBO = ON the device will be programmed for high speed; with TURBO = OFF the device will be programmed for automatic standby (power savings). The default state is OFF.

PACKAGING

Intel EPLDs are available in several packages to meet the wide requirements of customer applications. Current information on available packages is available from your local Intel field sales engineer. Detailed information on package dimensions, etc. for a particular package is provided in *Packaging Outlines and Dimensions*, Order Number 321369, which covers all Intel packages.

ORDERING INFORMATION

Intel EPLDs are identified as follows:


M D 5 C X X X — S S
Device
Speed
Technology
C — CHMOS
AC — Advanced CHMOS
Package Type
A — Hermetic, Pin Grid Array
D — Hermetic, Type D (Cerdip) Dip
N — Plastic, Leaded Chip Carrier
CJ — Ceramic, J Leaded Chip Carrier
P — Plastic Dip and Plastic Flatpack
R — Hermetic, Leadless Chip Carrier
X — Unpackaged Device

A — Indicates automotive operating temperature range (−40°C to +125°C)

J — Indicates a JAN qualified device, but is for internal identification purposes only. All JAN devices must be ordered by M38510 part number. (Example: M38510/42001 BQB), and will be marked in accordance with MIL-M-38510 specifications.

L — Indicates extended operating temperature range (−40°C to +85°C) express product with 160 + 8 hrs. dynamic burn-in.

*M — Indicates military operating temperature range (−55°C to +125°C)

Q — Indicates commercial temperature range (0°C to 70°C) express product with 160 + 8 hrs. dynamic burn-in.

T — Indicates extended temperature range (−40°C to +85°C) express product without burn-in.

— No letter indicates commercial temperature range (0°C to 70°C) without burn-in.

Examples:

QD5C060-45 Commercial with burn-in, ceramic Dip, 060 (600 gate) device, 45 nanosecond.

*On military temperature devices, B suffix indicates MIL-STD-883C level B processing.

EPLDs Erasable Programmable Logic Devices

2

2 EPLDs Erasable Programmable Logic Devices

DATA SHEET SPECIFICATIONS

The specifications in these data sheets reflect some changes in comparison to earlier data sheets. These changes were made to provide more accurate and usable information concerning Intel EPLDs. A summary of the changes follows.

D.C. Characteristics

I_{SB} Standby Current (formerly called I_{CC1}).

I_{CC} Operating Current (formerly called I_{CC2}). Test conditions have been specified in greater detail.

A.C. Characteristics (Synchronous)

f_{MAX} Maximum Frequency (new spec.). Maximum frequency operation with no signals fed back to other macrocells.

f_{CNT} Maximum Counting Frequency (formerly called f_1). Maximum frequency operation with some signals fed back to other macrocells.

t_{CO} Output Register Valid from CLK (formerly called t_{CO1}).

t_{CNT} Register Output Feedback to Register Input — Internal Path (formerly called t_{P1}).

A.C. Characteristics (Asynchronous)

f_{AMAX} Maximum Frequency (new spec.). Maximum frequency operation with no signals fed back to other macrocells.

f_{ACNT} Maximum Counting Frequency (formerly called f_{A1}). Maximum frequency operation with some signals fed back to other macrocells.

t_{ACO} Output Register Valid from CLK (formerly called t_{ACO1}).

t_{ACNT} Register Output Feedback to Register Input — Internal Path (formerly called t_{AP1}).

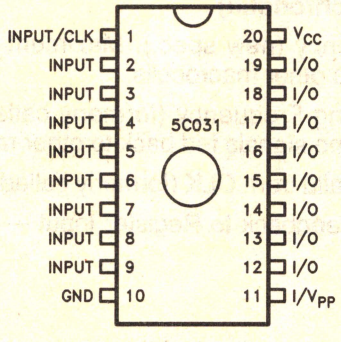
Non-Turbo Mode

The Non-Turbo Mode column in several of the data sheets shows the additional time required to power-up the device from standby mode. The column applies only when the device is operated in non-turbo mode (Turbo Bit Off) in an application where the device enters standby mode. See "Automatic Standby-Mode" in the Overview for additional information.

000274-1

5C031
300 GATE CHMOS H-SERIES ERASABLE PROGRAMMABLE LOGIC DEVICE (H-EPLD)

- **High Performance, Low Power Replacement for SSI & MSI Devices and Bipolar PLDs.**
- **Eight Macrocells with Programmable I/O Architecture.**
- **100% Generically Testable EPROM Logic Control Array.**
- **High Performance Upgrade for All Commonly Used 20-pin PLDs.**
- **CHMOS EPROM Technology Based UV Erasable.**
- **Up to 18 Inputs (10 Dedicated & 8 I/O) and 8 Outputs.**
- **Programmable "Security Bit" Allows Total Protection of Proprietary Designs**
- **I_{CC} (standby) 35 mA (max)**
  I_{CC} (10 MHz) 40 mA (max)
- **t_{PD} = 40 ns (max)**
- **20-pin 0.3" Windowed CERDIP Package**
  (See Packaging Spec., Order # 231369)



Pin Configuration

November 1987
Order Number: 290154-001

The Intel 5C031 H-EPLD (H-series Erasable Programmable Logic Device) is capable of implementing over 300 equivalent gates of user-customized logic functions through programming. This device can be used to replace bipolar programmable logic arrays and LS TTL and 74HC (CMOS) SSI and MSI logic devices. The 5C031 can also be used as a direct, low-power replacement for almost all common 20-pin fuse-based programmable logic devices. With its flexible programmable I/O architecture, this device has advanced functional capabilities beyond that of typical programmable logic.

The 5C031 H-EPLD uses CHMOS EPROM (floating gate) cells as logic control elements instead of fuses. The CHMOS EPROM technology reduces power consumption of H-EPLDs to less than 20% of a comparable bipolar device without sacrificing speed performance. In addition, the use of Intel's advanced CHMOS II-E EPROM process technology enables greater logic densities to be achieved with superior speed and low-power performance over other comparable devices. EPROM technology allows these devices to be 100% factory tested by programming and erasing all the EPROM logic control elements.

The 5C031 is housed in a windowed 0.3″ 20-pin DIP and has the benefits of being an ideal prototyping tool with its highly flexible I/O architecture.

ARCHITECTURE DESCRIPTION

The architecture of the 5C031 is based on the "Sum of Products" PLA (Programmable Logic Array) structure with a programmable AND array feeding into a fixed OR array. This device can accommodate both combinational and sequential logic functions. A proprietary programmable I/O architecture provides individual selection of either combinational or registered output and feedback signals, all with selectable polarity.

The 5C031 contains 10 dedicated inputs as well as 8 input/output pins. These I/O pins can be individually configured to be inputs, outputs or bi-directional I/O pins. Each of these I/O pins is connected to a macrocell. The 5C031 contains 8 identical macrocells organized as shown in Figure 1.

Each macrocell (see Figure 2) consists of a PLA (programmable logic array) block and an I/O architecture block, which contains a "D" type register. The PLA block consists of eight 36-input AND gates (TRUE & COMPLEMENT of 10 dedicated inputs plus the 8 feedback inputs from the eight macrocells), feeding into an OR gate. The output of this PLA block is fed into the I/O architecture block. The different I/O and feedback options that are achievable from the 5C031 I/O block are shown in Figure 3.

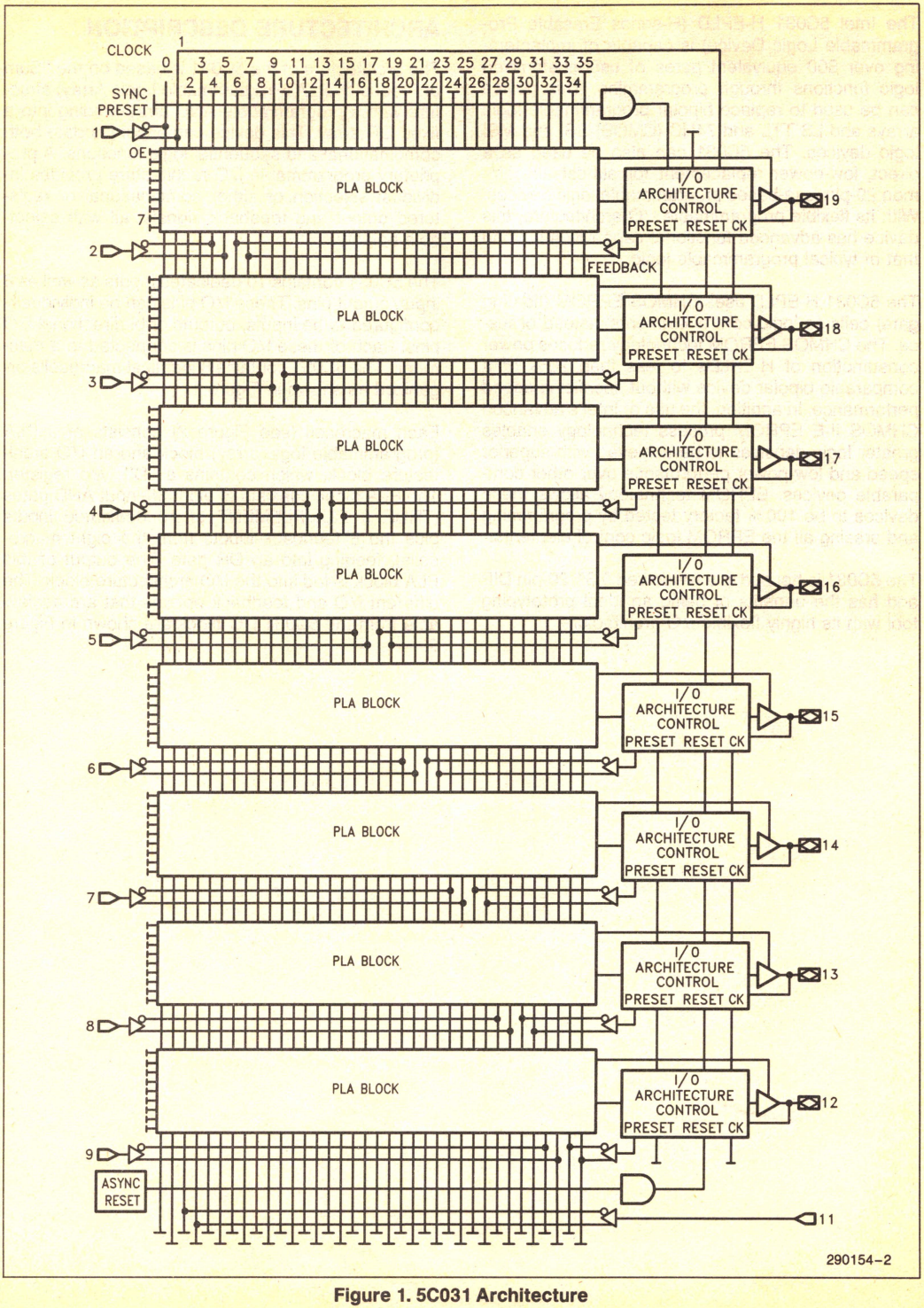


Figure 1. 5C031 Architecture

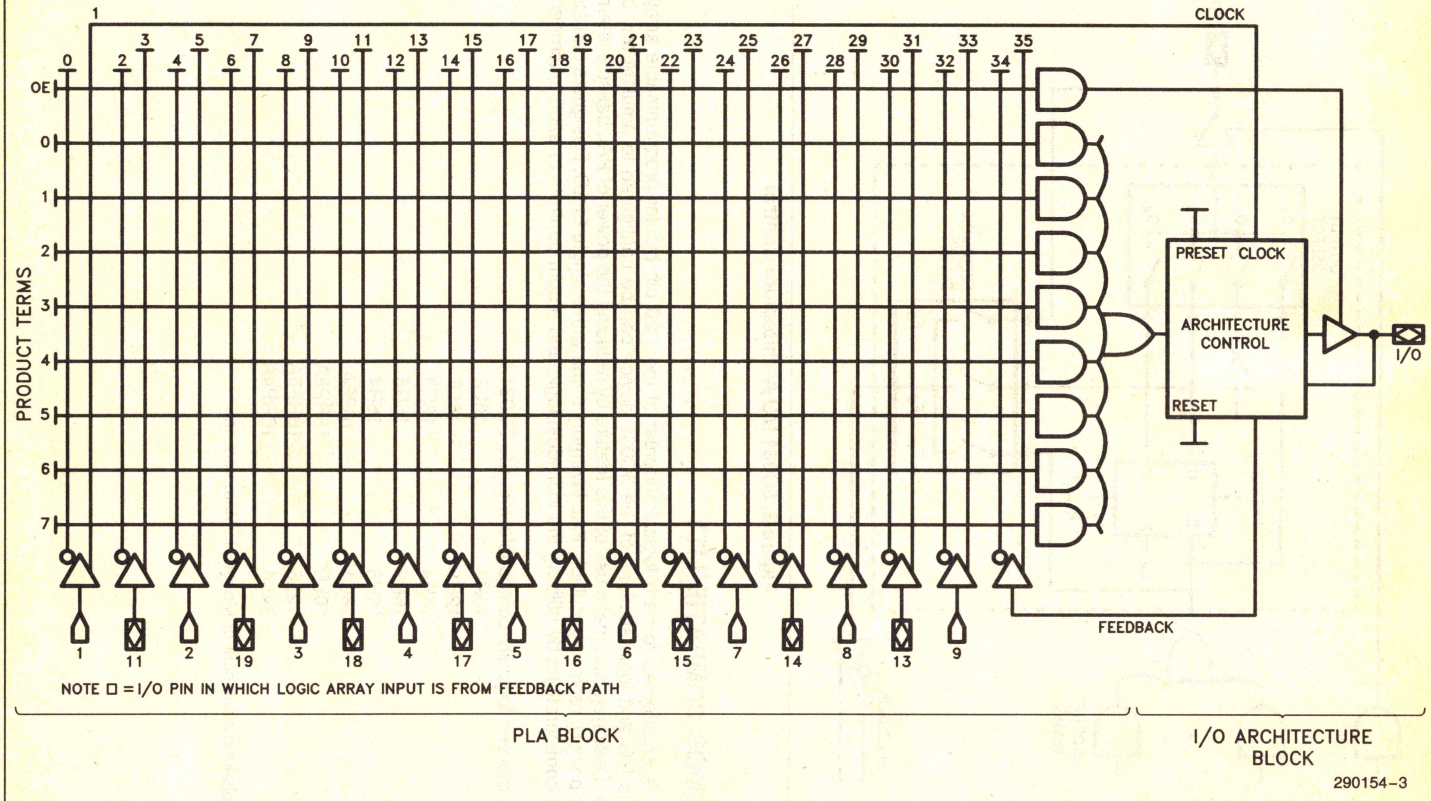

1
CLOCK
0 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
OE
0
1
2
3
4
5
6
7
PRODUCT TERMS
PRESET CLOCK
ARCHITECTURE CONTROL
RESET
I/O
FEEDBACK
1 11 2 19 3 18 4 17 5 16 6 15 7 14 8 13 9
NOTE □ = I/O PIN IN WHICH LOGIC ARRAY INPUT IS FROM FEEDBACK PATH
PLA BLOCK
I/O ARCHITECTURE BLOCK
290154-3

Figure 2. Logic Array Macrocell

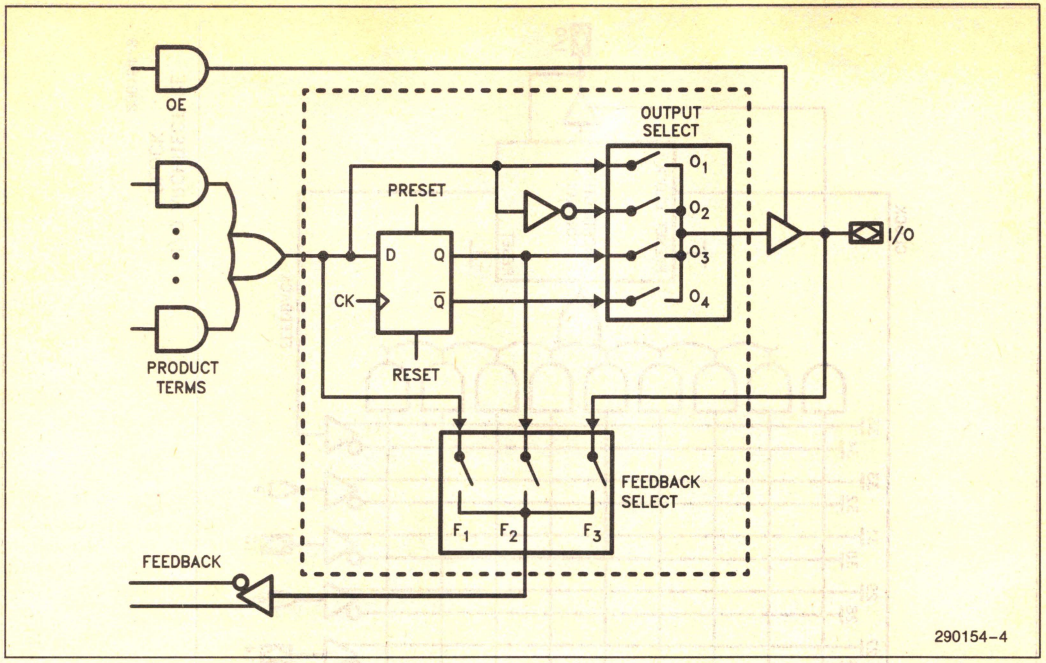


Figure 3. 5C031 I/O Architecture Control

20 PIN CMOS COMPATIBILITY

The 5C031 is architected to be a logical superset of most 20 pin bipolar programmable array logic (PAL*) devices. The I/O and logic sections of the 5C031 device can be configured to emulate any of the devices listed below. Designers can make use of this feature by reducing the power of PAL based systems (EPLDs are much lower power), replacing multiple PAL inventory items with a single EPLD. Designers can also create new 20 pin PLD configurations by utilizing the individual logic and output controls of each macrocell.

List of PAL devices logically compatible with the 5C031.

10H8	16L2
12H6	16L8
14H4	16R8
16H2	16R6
16H8	16R4
16C1	16P8A
10LB	16RP8A
12L6	16RP6A
14L4	16RP4A

*PAL is a registered trademark of Monolithic Memories, Inc.

Erased-State Configuration

Prior to programming or after erasing, the I/O structure is configured for combinatorial active low output with input (pin) feedback.

ERASURE CHARACTERISTICS

Erasure characteristics of the 5C031 are such that erasure begins to occur upon exposure to light with wavelengths shorter than approximately 4000Å. It should be noted that sunlight and certain types of flourescent lamps have wavelengths in the 3000–4000Å. Data shows that constant exposure to room level flourescent lighting could erase the typical 5C031 in approximately three years, while it would take approximately one week to cause erasure when exposed to direct sunlight. If the 5C031 is to be exposed to these types of lighting conditions for extended periods of time, conductive opaque labels should be placed over the device window to prevent unintentional erasure.

The recommended erasure procedure for the 5C031 is exposure to shortwave ultraviolet light with a wavelength of 2537Å. The integrated dose (i.e., UV intensity × exposure time) for erasure should be a minimum of fifteen (15) Wsec/cm^2. The erasure time with this dosage is approximately 15 to 20 minutes using an ultraviolet lamp with a 12,000 μW/cm^2 power rating. The 5C031 should be placed within one inch of the lamp tubes during erasure. The maximum integrated dose the 5C031 can be exposed to without damage is 7258 Wsec/cm^2 (1 week at 12,000 μW/cm^2). Exposure to high intensity UV light for longer periods may cause permanent damage to the device.

PROGRAMMING CHARACTERISTICS

Initially, and after erasure, all the EPROM control bits of the 5C031 are connected (in the "1" state). Each of the connected control bits are selectively disconnected by programming the EPROM cells into their "0" state. Programming voltage and waveform specifications are available by request from Intel to support programming of the 5C031.

int$_e$ligent Programming™ Algorithm

The 5C031 supports the int$_e$ligent Programming Algorithm which rapidly programs Intel H-ELPDs (and EPROMs) using an efficient and reliable method. The int$_e$ligent Programming Algorithm is particularly suited to the production programming environment. This method greatly decreases the overall programming time while programming reliability is ensured as the incremental program margin of each bit is continually monitored to determine when the bit has been successfully programmed.

FUNCTIONAL TESTING

Since the logical operation of the 5C031 is controlled by EPROM elements, the device is completely testable. Each programmable EPROM bit controlling the internal logic is tested using application-independent test program patterns. After testing, the devices are erased before shipment to customers. No post-programming tests of the EPROM array are required.

The testability and reliability of EPROM-based programmable logic devices is an important feature over similar devices based on fuse technology. Fuse-based programmable logic devices require a user to perform post-programming tests to insure proper programming. These tests must be done at the device level because of the cummulative error effect. For example, a board containing ten devices each possessing a 2% device fallout translates into an 18% fallout at the board level (it should be noted that programming fallout of fuse-based programmable logic devices is typically 2% or higher).

DESIGN RECOMMENDATIONS

To take maximum advantage of EPLD technology, it is recommended that the designer use the Modular EPLD Logic Design (MELD) method. The MELD philosophy is derived from the modular programming method used in software development. In a modular software development environment, the engineer designs a modular program (typically on a development system), stores it in memory (EPROM), and tests the module for functionality. A hardware designer using EPLDs can use this same approach when designing logic. The designer develops a modular logic design on the Intel Programmable Logic Development System II (iPLDS II), stores it in "memory" (the EPROM control elements of the EPLD), and again tests the module for functionality. If the design is in error, the logic designer reprograms the EPLD with his new design as easily as a software designer can download a new program into memory.

The MELD philosophy is new to programmable logic because EPROM-based PLDs are new. A modular logic development process using fused-based PLDs would be wasteful since a fused-based device cannot be erased an re-used.

For proper operation, it is recommended that all input and output pins be constrained to the voltage range GND < (V_{IN} or V_{OUT}) < V_{CC}. Unused inputs should be tied to an appropriate logic level (e.g. either V_{CC} or GND) to minimize device power consumption. Reserved pins (as indicated in the iPLDS REPORT file) should be left floating (no connect) so that the pin can attain the appropriate logic level. A power supply decoupling capacitor of at least 0.2 μF must be connected directly between V_{CC} and GND pins of the device.

DESIGN SECURITY

A single EPROM bit provides a programmable design security feature that controls the access to the data programmed into the device. If this bit is set, a proprietary design within the device cannot be copied. This EPROM security bit enables a higher degree of design security than fused-based devices since programmed data within EPROM cells is invisible even to microscopic evaluation. The EPROM security bit, along with all the other EPROM control bits, will be reset by erasing the device.

LATCH-UP IMMUNITY

All of the input, I/O, and clock pins of the 5C031 have been designed to resist latch-up which is inherent in inferior CMOS structures. The 5C031 is designed with Intel's proprietary CHMOS II-E EPROM process. Thus, each of the 5C031 pins will not experience latch-up with currents up to 100 mA and voltages ranging from -1V to V_{CC} + 1V. Furthermore, the programming pin is designed to resist latch-up to the 13.5V maximum device limit.

INTEL PROGRAMMABLE LOGIC DEVELOPMENT SYSTEM II (iPLDS II)

The iPLDS II graphically shown in Figure 5 provides all the tools needed to design with Intel H-Series EPLDs or compatible devices. In addition to providing development assistance, iPLDS II insulates the user from having to know all the intricate details of EPLD architecture (the machine will optimize a design to benefit from architectual features). It contains comprehensive third generation software that supports four different design entry methods, minimizes logic, does automatic pin assignments and produces the best design fit for the selected EPLD. It is user friendly with guided menus, on-line Help messages and soft key inputs.

In addition, the iPLDS II contains programmer hardware in the form of an iUP-PC Universal Programmer-Personal Computer to enable the user to program EPLDs, read and verify programmed devices and also to graphically edit programming files. The software generates industry standard JEDEC object code output files which can be downloaded to other programmers as well.

The iPLDS II has interfaces to popular schematic capture packages (including Dash series from FutureNet* and PC CAPS from PCAD)** to enable designs to be entered using schematics. A more integrated schematic entry method is provided by SCHEMA II-PLD, a low-cost schematic capture package that supports EPLD primitives and user-defined macro symbols. SCHEMA II-PLD contains the EPLD Design Manager, which provides a single user interface to both SCHEMA II-PLD and iPLS II software. The other design formats supported are Boolean equation entry and State Machine design entry.

The iPLDS operates on the IBM† PC/XT, PC/AT, or other compatible machine with the following configuration:

1. At least one floppy disk drive and hard disk drive.
2. MS-DOS†† Operating System Version 3.0 or greater.
3. 640K Memory.
4. Intel iUP-PC Universal Programmer-Personal Computer and GUPI Adaptor (supplied with iPLDS).
5. A color monitor is suggested.

Detailed information on the Intel Programmable Logic Development System II is contained in a separate Intel data sheet. (Order Number: 280168)

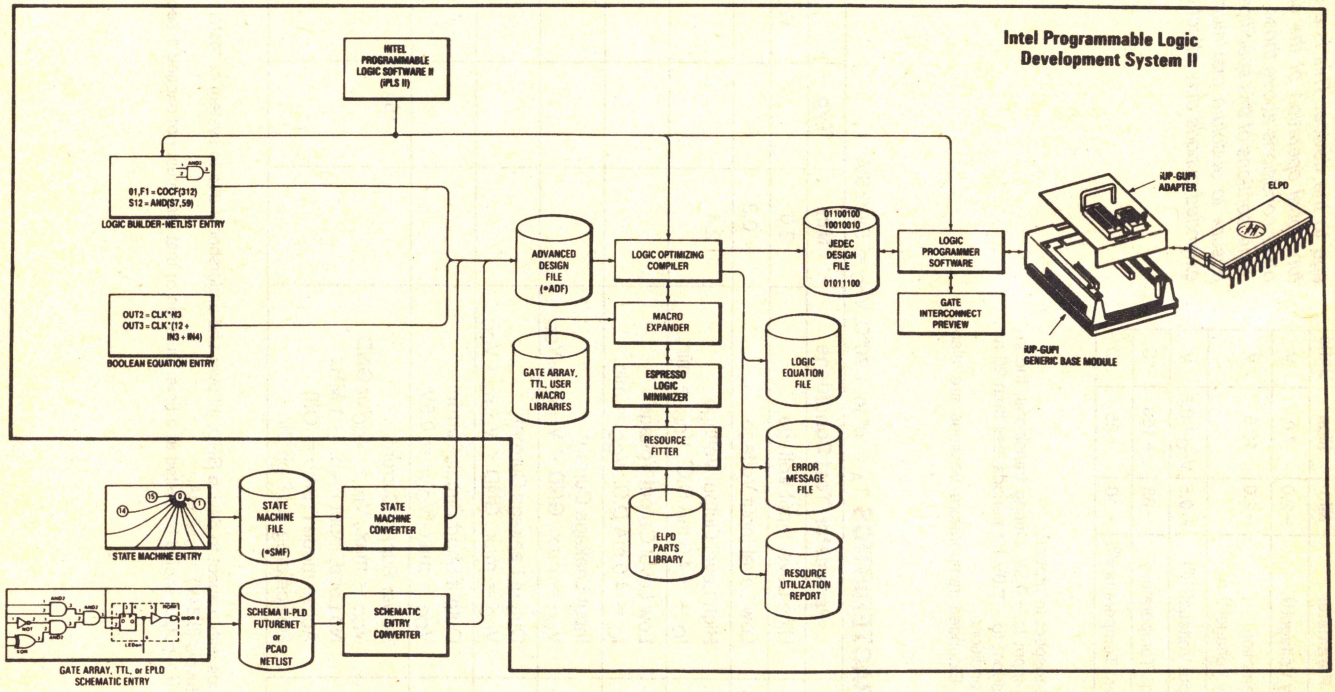
*FutureNet is a registered trademark of FutureNet Corporation. DASH is a trademark of FutureNet Corporation.

**PC-CAPS is a trademark of P-CAD Corporation.

†IBM Personal Computer is a registered trademark of International Business Machines Corporation.

††MS-DOS is a registered trademark of Microsoft Corporation.

iPLDS INTEL PROGRAMMABLE LOGIC DEVELOPMENT SYSTEM



290154-5

Figure 5. iPLDS II Intel Programmable Logic Development System

ABSOLUTE MAXIMUM RATINGS*

Symbol	Parameter	Min	Max	Units
V_{CC}	Supply Voltage(1)	−2.0	7.0	V
V_{PP}	Programming Supply Voltage(1)	−2.0	13.5	V
V_I	DC Input Voltage(1)(2)	−0.5	V_{CC}+0.5	V
t_{stg}	Storage Temperature	−65	+150	°C
t_{amb}	Ambient Temperature(3)	−10	+85	°C

NOTES:
1. Voltages with respect to ground.
2. Minimum DC input is −0.5V. During transitions, the inputs may undershoot to −2.0V for periods less than 20 ns under no load conditions.
3. Under bias. Extended temperature versions are also available.

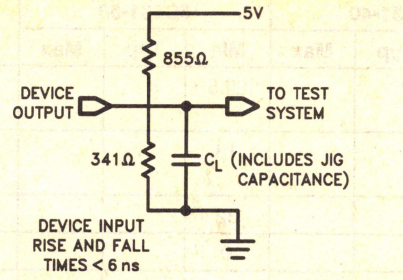
**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS T_A = 0° to +70°C, V_{CC} = 5V ±5%

Symbol	Parameter/Test Conditions	Min	Typ	Max	Unit
V_{IH}(4)	High Level Input Voltage	2.0		V_{CC} + 0.3	V
V_{IL}(4)	Low Level Input Voltage	−0.3		0.8	V
V_{OH}(5)	High Level Output Voltage I_O = −4.0 mA D.C., V_{CC} = min.	2.4			V
V_{OL}	Low Level Output Voltage I_O = 4.0 mA D.C., V_{CC} = min.			0.45	V
I_I	Input Leakage Current V_{CC} = max., GND < V_{OUT} < V_{CC}			±10	µA
I_{OZ}	Output Leakage Current V_{CC} = max., GND < V_{OUT} < V_{CC}			±10	µA
I_{SC}(6)	Output Short Circuit Current V_{CC} = max., V_{OUT} = 0.5V			10	mA
I_{CC}	Power Supply Current V_{CC} = max., V_{IN} = V_{CC} or GND No Load, Input Freq. = 1 MHz Active mode (Turbo = Off) Device prog. as 8-bit Ctr.		15	40	mA

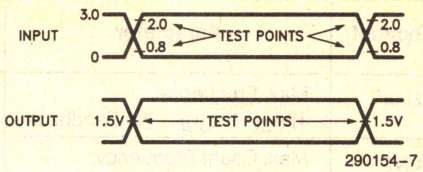
NOTES:
4. Absolute values with respect to device GND; all over and undershoots due to system or tester noise are included.
5. I_O at CMOS levels (3.84V) = −2 mA.
6. Not more than 1 output should be tested at a time. Duration of that test must not exceed 1 second.

A.C. TESTING LOAD CIRCUIT



C_L = 50 pF

A.C. TESTING INPUT, OUTPUT WAVEFORM



A.C. Testing: Inputs are Driven at 3.0V for a Logic "1" and 0V for a Logic "0". Timing Measurements are made at 2.0V for a Logic "1" and 0.8V for a Logic "0" on inputs. Outputs are measured at a 1.5V point.

A.C. CHARACTERISTICS T_A = 0°C to +70°C, V_{CC} = 5V ±5%, Turbo Bit Programmed(7)

Symbol	From	To	5C031-40			5C031-50			Unit
			Min	Typ	Max	Min	Typ	Max	
t_{PD}	I/O	Comb. Output			40			50	ns
t_{PZX}(8)	I or I/O	Output Enable			40			50	ns
t_{PXZ}(8)	I or I/O	Output Disable			40			50	ns
t_{CLR}	Asynch Reset	Q Reset			40			50	ns

NOTES:
7. Typical Values are at T_A = 25°C, V_{CC} = 5V, Active Mode
8. t_{PZX} and t_{PXZ} are measured at ±0.5V from steady state voltage as driven by spec. output load. t_{PXZ} is measured with C_L = 5 pF.

CAPACITANCE

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
C_{IN}	Input Capacitance	V_{IN} = 0V, f = 1.0 MHz			20	pF
C_{OUT}	Output Capacitance	V_{OUT} = 0V, f = 1.0 MHz			20	pF
C_{CLK}	Clock Pin Capacitance	V_{OUT} = 0V, f = 1.0 MHz			20	pF
C_{VPP}	V_{PP} Pin	Pin 11			50	pF

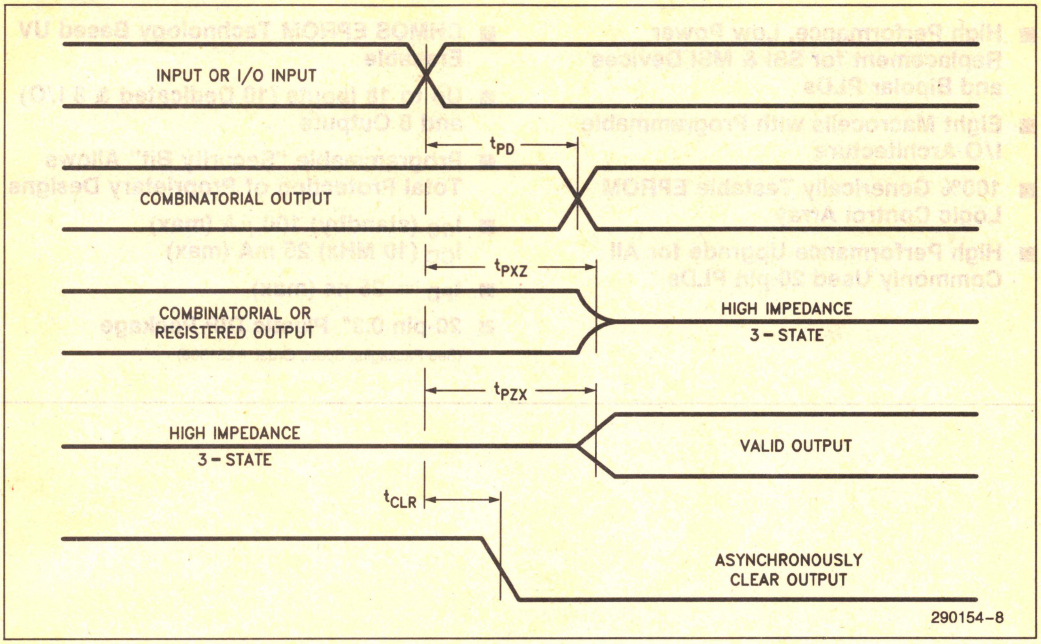
SYNCHRONOUS CLOCK MODE A.C. CHARACTERISTICS

T_A = 0°C to +70°C, V_{CC} = 5.0V ±5%, Turbo Bit On[7]

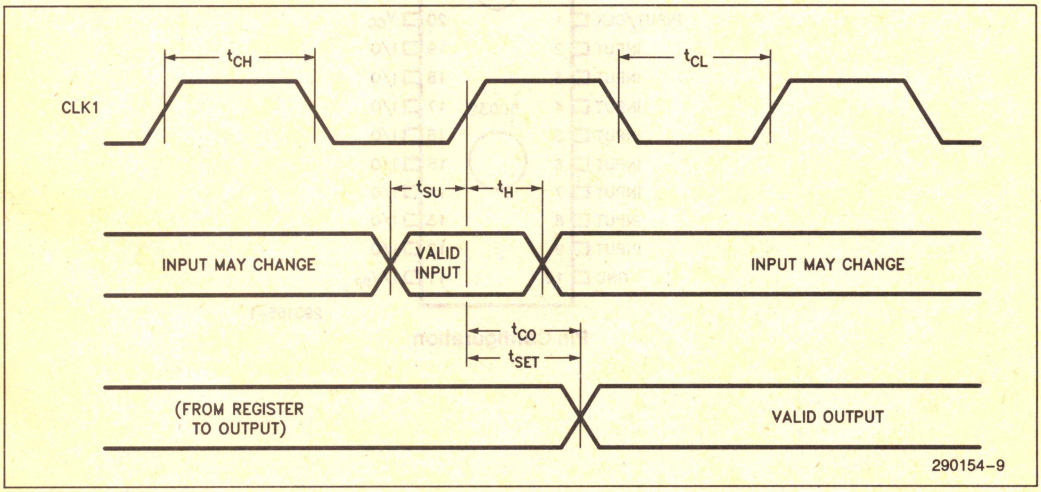
Symbol	Parameter	5C031-40			5C031-50			Unit
		Min	Typ	Max	Min	Typ	Max	
f_{MAX}	Max. Frequency $1/(t_{CL} + t_{CH})$— No Feedback	29.5			22.5			MHz
f_{CNT}	Max. Count Frequency $1/t_{CNT}$ — With Feedback	22			18			MHz
t_{SU}	I/O Setup Time to CLK	30			32			ns
t_H	I or I/O Hold after CLK High	0			0			ns
t_{CO}	CLK High to Output Valid			24			28	ns
t_{CNT}	Register Output Feedback to Register Input — Internal Path			45			55	ns
t_{CH}	CLK High Time	17			22			ns
t_{CL}	CLK Low Time	17			22			ns
t_{SET}	Synch. Set to Q Set			40			50	ns

SWITCHING WAVEFORMS

COMBINATORIAL MODE

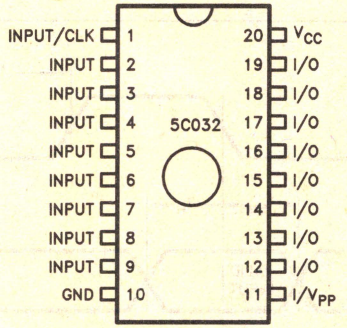


SYNCHRONOUS CLOCK MODE



5C032
300 GATE CHMOS H-SERIES ERASABLE PROGRAMMABLE LOGIC DEVICE (H-EPLD)

- High Performance, Low Power Replacement for SSI & MSI Devices and Bipolar PLDs
- Eight Macrocells with Programmable I/O Architecture
- 100% Generically Testable EPROM Logic Control Array
- High Performance Upgrade for All Commonly Used 20-pin PLDs
- CHMOS EPROM Technology Based UV Erasable
- Up to 18 Inputs (10 Dedicated & 8 I/O) and 8 Outputs
- Programmable "Security Bit" Allows Total Protection of Proprietary Designs
- I_{CC} (standby) 100 μA (max)
  I_{CC} (10 MHz) 25 mA (max)
- t_{PD} = 25 ns (max)
- 20-pin 0.3" Plastic DIP Package
  (See Packaging Spec., Order #231369)



290155-1

Pin Configuration

November 1987
Order Number: 290155-001

The Intel 5C032 H-EPLD (H-series Erasable Programmable Logic Device) is capable of implementing over 300 equivalent gates of user-customized logic functions through programming. This device can be used to replace bipolar programmable logic arrays and LS TTL and 74HC (CMOS) SSI and MSI logic devices. The 5C032 can also be used as a direct, low-power replacement for almost all common 20-pin fuse-based programmable logic devices. With its flexible programmable I/O architecture, this device has advanced functional capabilities beyond that of typical programmable logic.

The 5C032 H-EPLD uses CHMOS EPROM (floating gate) cells as logic control elements instead of fuses. The CHMOS EPROM technology reduces power consumption of H-EPLDs to less than 20% of a comparable bipolar device without sacrificing speed performance. In addition, the use of Intel's advanced CHMOS II-E EPROM process technology enables greater logic densities to be achieved with superior speed and low-power performance over other comparable devices. Intel's 5C032 has the benefit of "zero" stand-by power not available on other programmable logic devices. EPROM technology allows these devices to be 100% factory tested by programming and erasing all the EPROM logic control elements.

The 5C032 with its superior speed and power performance and its plastic package is an ideal production vehicle for high-volume manufacturing. Most commonly used 20-pin bipolar PLDs can be easily replaced with this device allowing for tremendous power consumption savings without sacrificing speed of operation.

ARCHITECTURE DESCRIPTION

The architecture of the 5C032 is based on the "Sum of Products" PLA (Programmable Logic Array) structure with a programmable AND array feeding into a fixed OR array. This device can accommodate both combinational and sequential logic functions. A proprietary programmable I/O architecture provides individual selection of either combinational or registered output and feedback signals, all with selectable polarity.

The 5C032 contains 10 dedicated inputs as well as 8 input/output pins. These I/O pins can be individually configured to be inputs, outputs or bi-directional I/O pins. Each of these I/O pins is connected to a macrocell. The 5C032 contains 8 identical macrocells organized as shown in Figure 1.

Each macrocell (see Figure 2) consists of a PLA (programmable logic array) block and an I/O architecture block, which contains a "D" type register. The PLA block consists of eight 36-input AND gates (TRUE & COMPLEMENT of 10 dedicated inputs plus the 8 feedback inputs from the eight macrocells), feeding into an OR gate. The output of this PLA block is fed into the I/O architecture block. The different I/O and feedback options that are available in the 5C032 I/O block are shown in Figure 3.

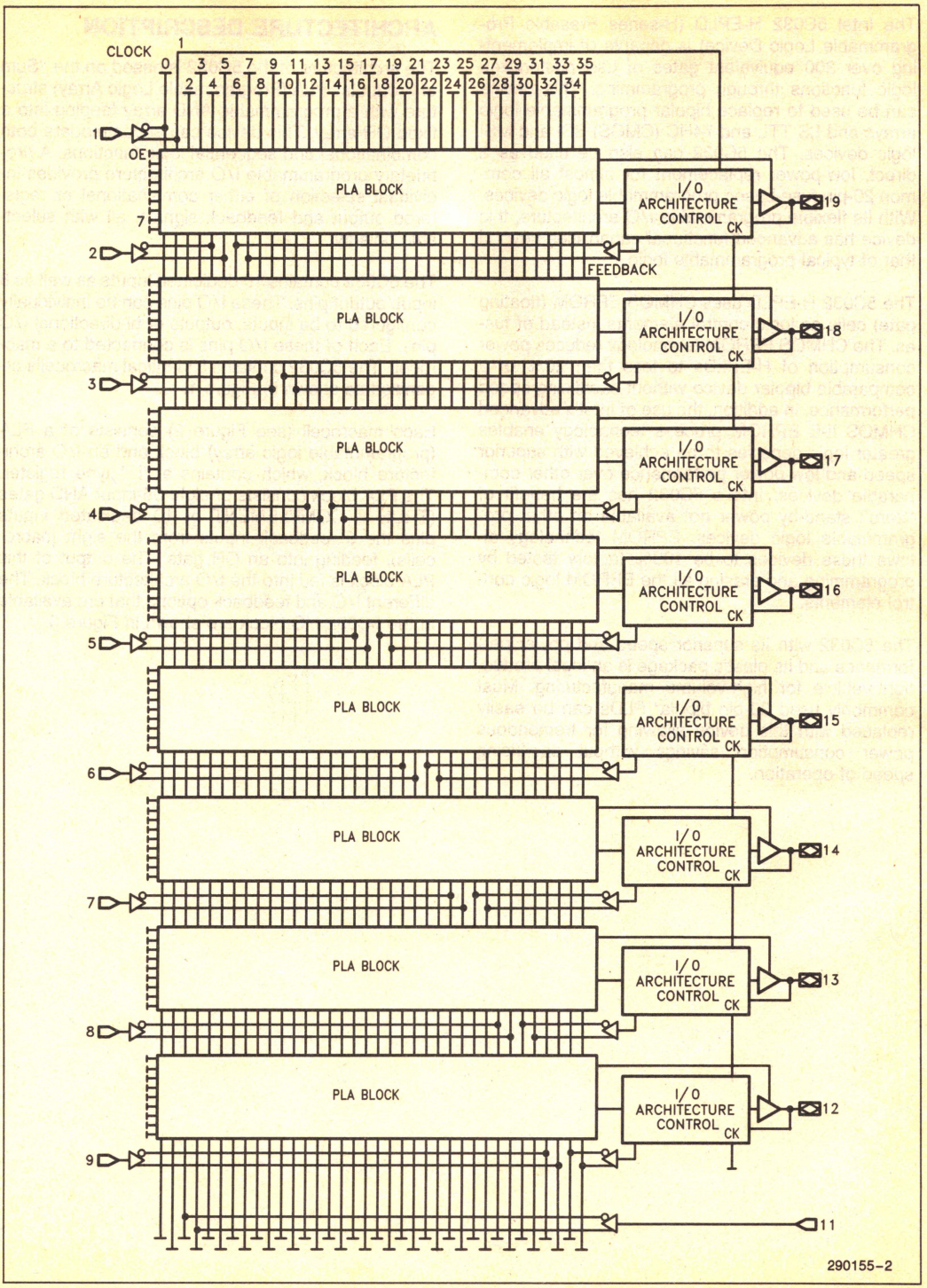


Figure 1. 5C032 Architecture

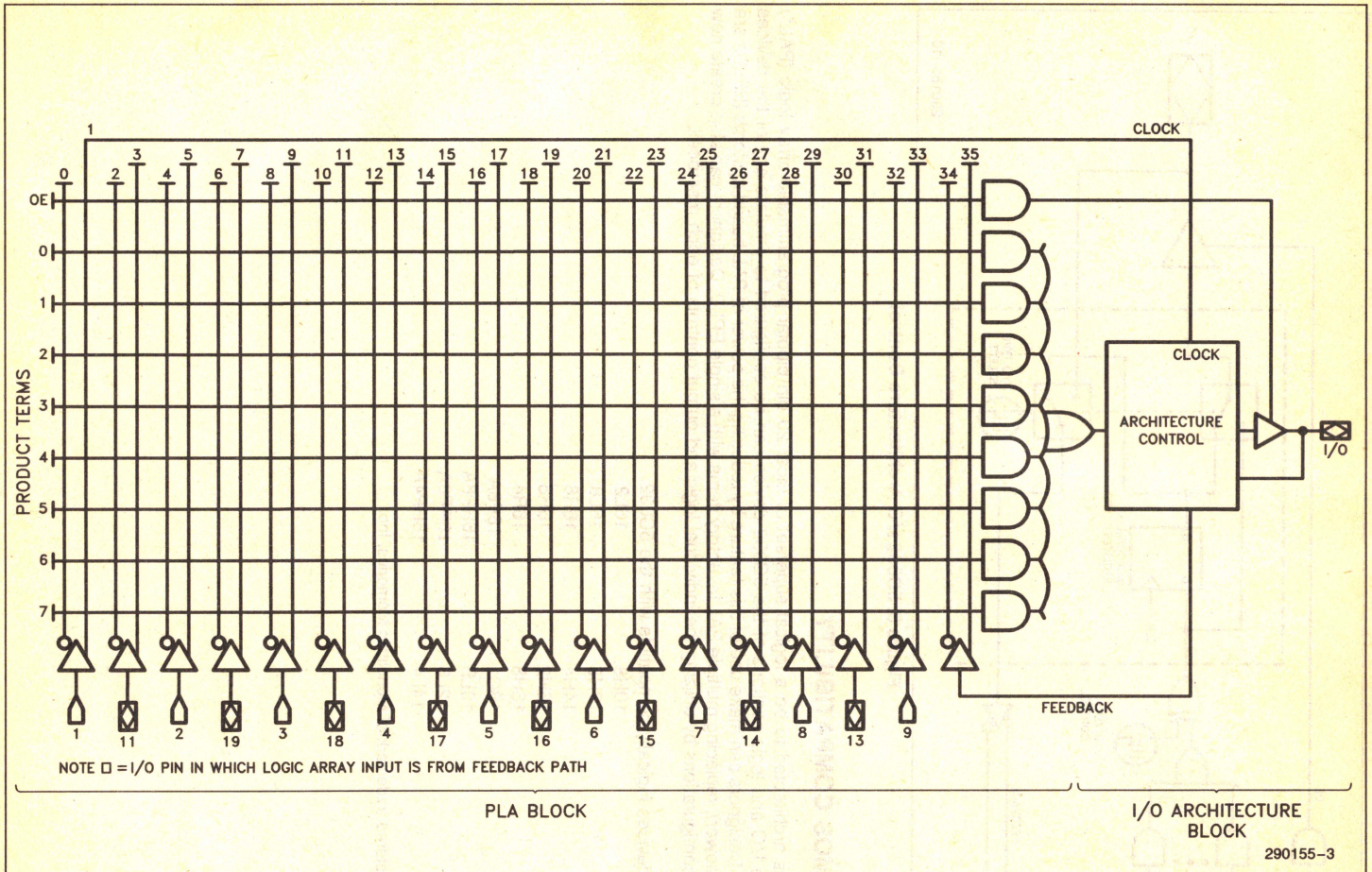

1
CLOCK
0 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
OE
0
1
2
3
4
5
6
7
PRODUCT TERMS
CLOCK
ARCHITECTURE CONTROL
I/O
FEEDBACK
1 11 2 19 3 18 4 17 5 16 6 15 7 14 8 13 9
NOTE □ = I/O PIN IN WHICH LOGIC ARRAY INPUT IS FROM FEEDBACK PATH
PLA BLOCK
I/O ARCHITECTURE BLOCK
290155-3

Figure 2. Logic Array Macrocell

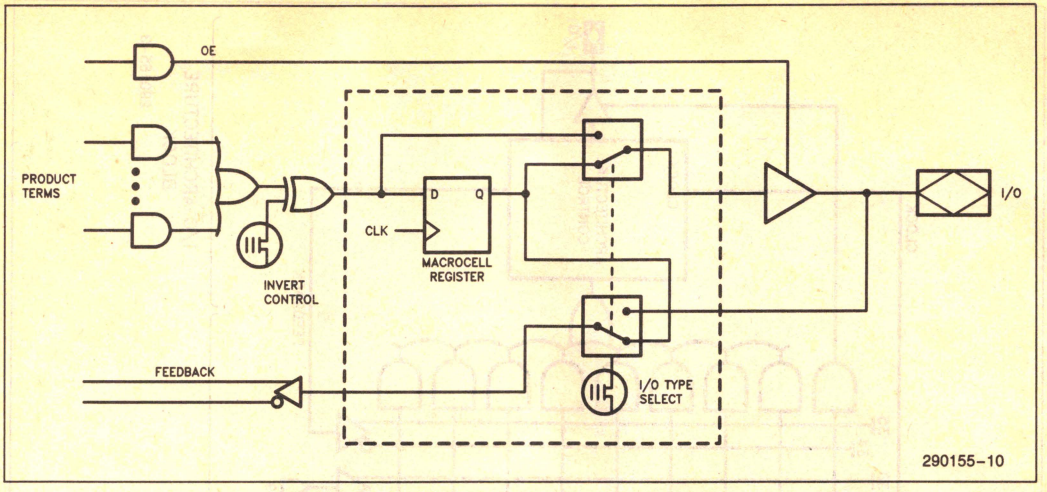


Figure 3. 5C032 I/O Architecture Control

20 PIN CMOS COMPATIBILITY

The 5C032 is architected to be a logical superset of most 20 pin bipolar programmable array logic (PAL*) devices. The I/O and logic sections of the 5C032 device can be configured to emulate any of the devices listed below. Designers can make use of this feature by reducing the power of PAL based systems (EPLDs are much lower power), replacing multiple PAL inventory items with a single EPLD. Designers can also create new 20 pin PLD configurations by utilizing the individual logic and output controls of each macrocell.

List of PAL devices logically compatible with the 5C032.

10H8	16L2
12H6	16L8
14H4	16R8
16H2	16R6
16H8	16R4
16C1	16P8A
10LB	16RP8A
12L6	16RP6A
14L4	16RP4A

*PAL is a registered trademark of Monolithic Memories, Inc.

Erased-State Configuration

Prior to programming or after erasing, the I/O structure is configured for combinatorial active low output with input (pin) feedback.

ERASURE CHARACTERISTICS

Erasure characteristics of the 5C032 are such that erasure begins to occur upon exposure to light with wavelengths shorter than approximately 4000Å. It should be noted that sunlight and certain types of flourescent lamps have wavelengths in the 3000–4000Å. Data shows that constant exposure to room level flourescent lighting could erase the typical 5C032 in approximately three years, while it would take approximately one week to cause erasure when exposed to direct sunlight. If the 5C032 is to be exposed to these types of lighting conditions for extended periods of time, conductive opaque labels should be placed over the device window to prevent unintentional erasure.

The recommended erasure procedure for the 5C032 is exposure to shortwave ultraviolet light with a wavelength of 2537Å. The integrated dose (i.e., UV intensity $\times$ exposure time) for erasure should be a minimum of fifteen (15) Wsec/cm^2. The erasure time with this dosage is approximately 15 to 20 minutes using an ultraviolet lamp with a 12,000 μW/cm^2 power rating. The 5C032 should be placed within one inch of the lamp tubes during erasure. The maximum integrated dose the 5C032 can be exposed to without damage is 7258 Wsec/cm^2 (1 week at 12,000 μW/cm^2). Exposure to high intensity UV light for longer periods may cause permanent damage to the device.

PROGRAMMING CHARACTERISTICS

Initially, and after erasure, all the EPROM control bits of the 5C032 are connected (in the "1" state). Each of the connected control bits are selectively disconnected by programming the EPROM cells into their "0" state. Programming voltage and waveform specifications are available by request from Intel to support programming of the device.

int$_e$ligent Programming™ Algorithm

The 5C032 supports the int$_e$ligent Programming Algorithm which rapidly programs Intel H-ELPDs (and EPROMs) using an efficient and reliable method. The int$_e$ligent Programming Algorithm is particularly suited to the production programming environment. This method greatly decreases the overall programming time while programming reliability is ensured as the incremental program margin of each bit is continually monitored to determine when the bit has been successfully programmed.

FUNCTIONAL TESTING

Since the logical operation of the 5C032 is controlled by EPROM elements, the device is completely testable. Each programmable EPROM bit controlling the internal logic is tested using application-independent test program patterns. After testing, the devices are erased before shipment to customers. No post-programming tests of the EPROM array are required.

The testability and reliability of EPROM-based programmable logic devices is an important feature over similar devices based on fuse technology. Fuse-based programmable logic devices require a user to perform post-programming tests to insure proper programming. These tests must be done at the device level because of the cummulative error effect. For example, a board containing ten devices each possessing a 2% device fallout translates into an 18% fallout at the board level (it should be noted that programming fallout of fuse-based programmable logic devices is typically 2% or higher).

DESIGN RECOMMENDATIONS

To take maximum advantage of EPLD technology, it is recommended that the designer use the Modular EPLD Logic Design (MELD) method. The MELD philosophy is derived from the modular programming method used in software development. In a modular software development environment, the engineer designs a modular program (typically on a development system), stores it in memory (EPROM), and tests the module for functionality. A hardware designer using EPLDs can use this same approach when designing logic. The designer develops a modular logic design on the Intel Programmable Logic Development System II (iPLDS II), stores it in "memory" (the EPROM control elements of the EPLD), and again tests the module for functionality. If the design is in error, the logic designer reprograms the EPLD with his new design as easily as a software designer can download a new program into memory.

The MELD philosophy is new to programmable logic because EPROM-based PLDs are new. A modular logic development process using fused-based PLDs would be wasteful since a fused-based device cannot be erased an re-used.

For proper operation, it is recommended that all input and output pins be constrained to the voltage range GND < (V_{IN} or V_{OUT}) < V_{CC}. Unused inputs should be tied to an appropriate logic level (e.g. either V_{CC} or GND) to minimize device power consumption. Reserved pins (as indicated in the iPLDS REPORT file) should be left floating (no connect) so that the pin can attain the appropriate logic level. A power supply decoupling capacitor of at least 0.2 μF must be connected directly between V_{CC} and GND pins of the device.

DESIGN SECURITY

A single EPROM bit provides a programmable design security feature that controls the access to the data programmed into the device. If this bit is set, a proprietary design within the device cannot be copied. This EPROM security bit enables a higher degree of design security than fused-based devices since programmed data within EPROM cells is invisible even to microscopic evaluation. The EPROM security bit, along with all the other EPROM control bits, will be reset by erasing the device.

LATCH-UP IMMUNITY

All of the input, I/O, and clock pins of the 5C032 have been designed to resist latch-up which is inherent in inferior CMOS structures. The 5C032 is designed with Intel's proprietary CHMOS II-E EPROM process. Thus, each of the 5C032 pins will not experience latch-up with currents up to 100 mA and voltages ranging from -1V to V_{CC} + 1V. Furthermore, the programming pin is designed to resist latch-up to the 13.5V maximum device limit.

INTEL PROGRAMMABLE LOGIC DEVELOPMENT SYSTEM II (iPLDS II)

The iPLDS II graphically shown in Figure 5 provides all the tools needed to design with Intel H-Series EPLDs or compatible devices. In addition to providing development assistance, iPLDS II insulates the user from having to know all the intricate details of EPLD architecture (the machine will optimize a design to benefit from architectual features). It contains comprehensive third generation software that supports four different design entry methods, minimizes logic, does automatic pin assignments and produces the best design fit for the selected EPLD. It is user friendly with guided menus, on-line Help messages and soft key inputs.

In addition, the iPLDS II contains programmer hardware in the form of an iUP-PC Universal Programmer-Personal Computer to enable the user to program EPLDs, read and verify programmed devices and also to graphically edit programming files. The software generates industry standard JEDEC object code output files which can be downloaded to other programmers as well.

The iPLDS II has interfaces to popular schematic capture packages (including Dash series from FutureNet* and PC CAPS from PCAD)** to enable designs to be entered using schematics. A more integrated schematic entry method is provided by SCHEMA II-PLD, a low-cost schematic capture package that supports EPLD primitives and user-defined macro symbols. SCHEMA II-PLD contains the EPLD Design Manager, which provides a single user interface to both SCHEMA II-PLD and iPLS II software. The other design formats supported are Boolean equation entry and State Machine design entry.

The iPLDS operates on the IBM† PC/XT, PC/AT, or other compatible machine with the following configuration:

1. At least one floppy disk drive and hard disk drive.
2. MS-DOS†† Operating System Version 3.0 or greater.
3. 640K Memory.
4. Intel iUP-PC Universal Programmer-Personal Computer and GUPI Adaptor (supplied with iPLDS II).
5. A color monitor is suggested.

Detailed information on the Intel Programmable Logic Development System II is contained in a separate Intel data sheet. (Order Number: 280168)

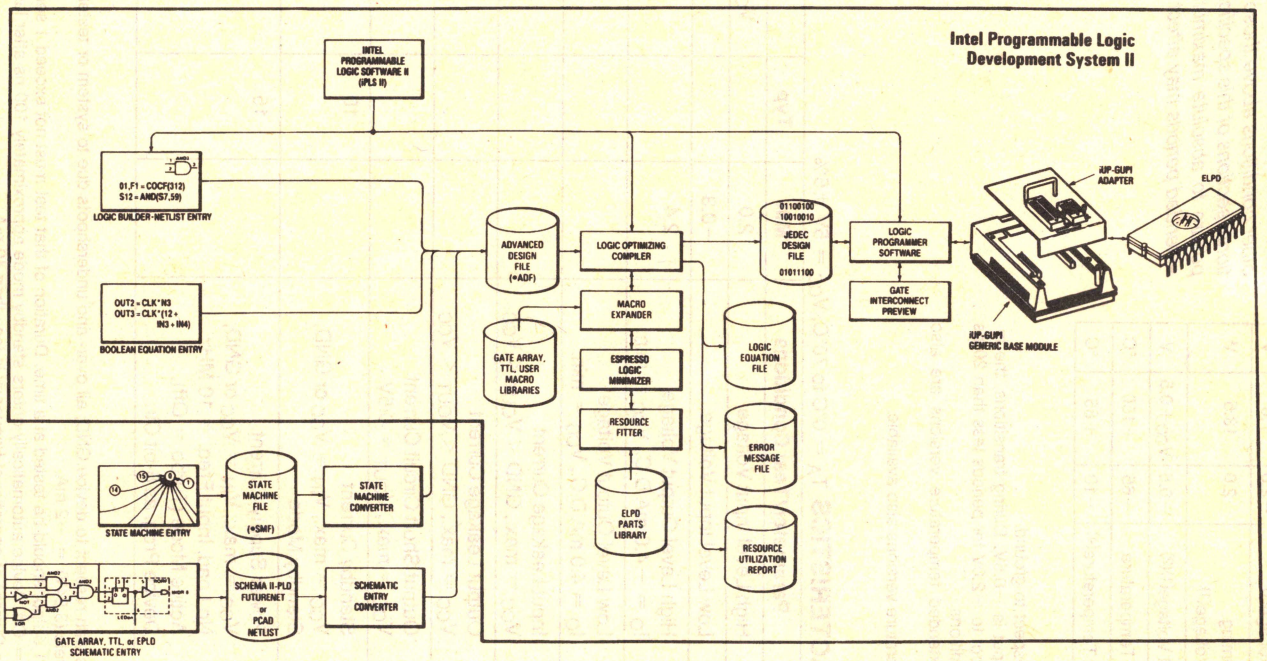
*FutureNet is a registered trademark of FutureNet Corporation. DASH is a trademark of FutureNet Corporation.

**PC-CAPS is a trademark of P-CAD Corporation.

†IBM Personal Computer is a registered trademark of International Business Machines Corporation.

††MS-DOS is a registered trademark of Microsoft Corporation.

iPLDS INTEL PROGRAMMABLE LOGIC DEVELOPMENT SYSTEM



290155-5

Figure 5. iPLDS II Intel Programmable Logic Development System

ABSOLUTE MAXIMUM RATINGS*

Symbol	Parameter	Min	Max	Unit
V_{CC}	Supply Voltage(1)	−2.0	7.0	V
V_{PP}	Programming Supply Voltage(1)	−2.0	13.5	V
V_I	DC Input Voltage(1)(2)	−0.5	V_{CC}+0.5	V
t_{stg}	Storage Temperature	−65	+150	°C
t_{amb}	Ambient Temperature(4)	−10	+85	°C

Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

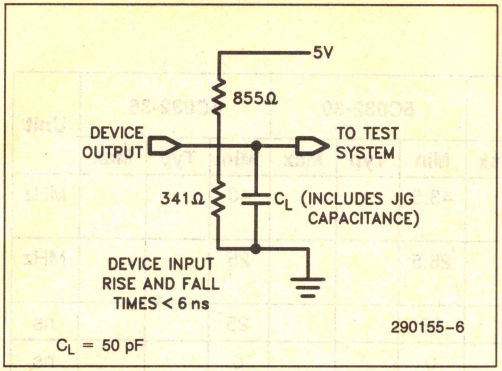
NOTES:
1. Voltages with respect to ground.
2. Minimum DC input is −0.5V. During transitions, the inputs may undershoot to −2.0V for periods less than 20 ns under no load conditions.
3. Under bias, Extended temperature versions are also available.
4. Extended temperature versions also available.

D.C. CHARACTERISTICS T_A = 0°C to 70°C, V_{CC} = 5V ±5%

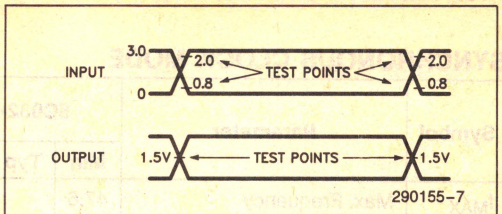
Symbol	Parameter/Test Conditions	Min	Typ	Max	Unit
V_{IH}(5)	High Level Input Voltage	2.0		V_{CC} + 0.3	V
V_{IL}(5)	Low Level Input Voltage	−0.3		0.8	V
V_{OH}(6)	High Level Output Voltage I_O = −4.0 mA D.C., V_{CC} = min.	2.4			V
V_{OL}	Low Level Output Voltage I_O = 4.0 mA D.C., V_{CC} = min.			0.45	V
I_I	Input Leakage Current V_{CC} = max., GND < V_{OUT} < V_{CC}			±10	µA
I_{OZ}	Output Leakage Current V_{CC} = max., GND < V_{OUT} < V_{CC}			±10	µA
I_{SC}(7)	Output Short Circuit Current V_{CC} = max., V_{OUT} = 0.5V			10	mA
I_{SB}(8)	Standby Current V_{CC} = max., V_{IN} = V_{CC} or GND, Standby Mode		10	100	µA
I_{CC}(9)	Power Supply Current V_{CC} = max., V_{IN} = V_{CC} or GND, No Load, Input Freq. = 10 MHz Active Mode (Turbo = Off), Device Prog. as 8-bit Ctr.		15	25	mA

NOTES:
5. Absolute values with respect to device GND; all over- and undershoots due to system or tester noise are included.
6. I_O at CMOS levels (3.84V) = −2 mA.
7. Not more than 1 output should be tested at a time. Duration of that test must not exceed 1 second.
8. With Turbo Bit = Off, device automatically enters standby mode approximately 100 ns after last input transition.
9. Maximum Active Current at operational frequency is less than 40 mA.

A.C. TESTING LOAD CIRCUIT



A.C. TESTING INPUT, OUTPUT WAVEFORM



A.C. Testing: Inputs are Driven at 3.0V for a Logic "1" and 0V for a Logic "0". Timing Measurements are made at 2.0V for a Logic "1" and 0.8V for a Logic "0" on inputs. Outputs are measured at a 1.5V point.

A.C. CHARACTERISTICS $T_A = 0°C$ to $+70°C$, $V_{CC} = 5V \pm 5\%$, Turbo Bit On(10)

Symbol	From	To	5C032-25			5C032-30			5C032-35			Unit
			Min	Typ	Max	Min	Typ	Max	Min	Typ	Max	
t_{PD}	I or I/O	Comb. Output			25			30			35	ns
t_{PZX}(11)	I or I/O	Output Enable			25			30			35	ns
t_{PXZ}(11)	I or I/O	Output Disable			25			30			35	ns

NOTES:
10. Typ. values are at $T_A = 25°C$, $V_{CC} = 5V$, Active Mode.
11. t_{PZX} and t_{PXZ} are measured at $\pm 0.5V$ from steady state voltage as driven by spec. output load. t_{PXZ} is measured with $C_L = 5$ pF.

CAPACITANCE

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
C_{IN}	Input Capacitance	$V_{IN} = 0V$, f = 1.0 MHz			20	pF
C_{OUT}	Output Capacitance	$V_{OUT} = 0V$, f = 1.0 MHz			20	pF
C_{CLK}	Clock Pin Capacitance	$V_{OUT} = 0V$, f = 1.0 MHz			20	pF
C_{VPP}(12)	V_{PP} Pin				50	pF

NOTE:
12. V_{PP} is on Pin 11.

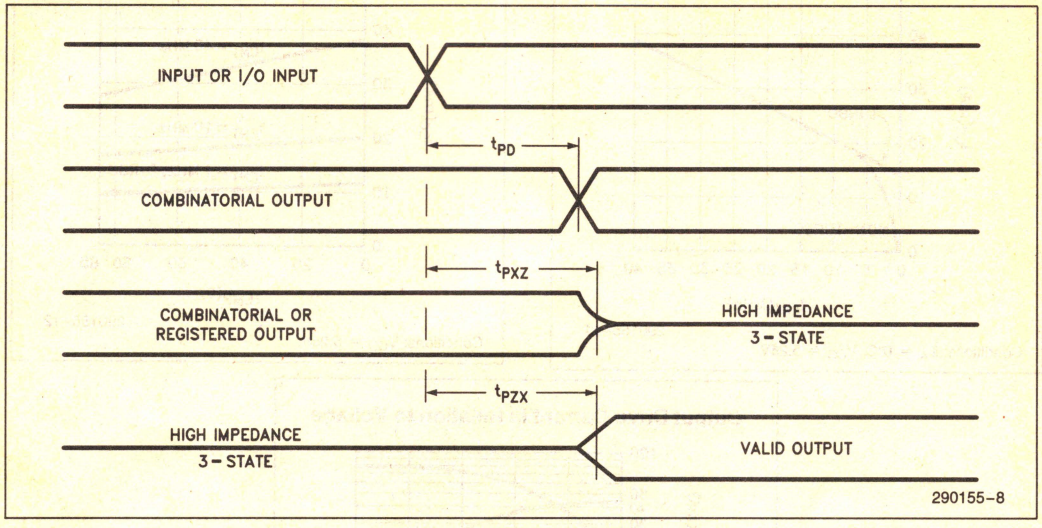
A.C. CHARACTERISTICS T_A = 0°C to 70°C, V_{CC} = 5V ±5%, Turbo Bit On (10)

SYNCHRONOUS CLOCK MODE

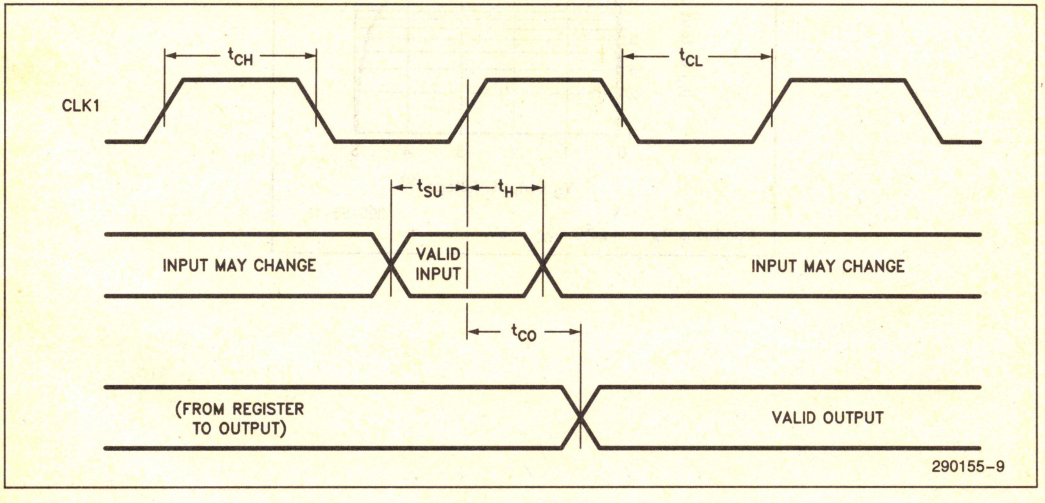
Symbol	Parameter	5C032-25			5C032-30			5C032-35			Unit
		Min	Typ	Max	Min	Typ	Max	Min	Typ	Max	
f_{MAX}	Max. Frequency $1/t_{SU}$ — No Feedback	47.6			43.5			40			MHz
f_{CNT}	Max. Count Frequency $1/t_{CNT}$ — with Feedback	33.3			28.5			25			MHz
t_{SU}	Input Setup Time to CLK	21			23			25			ns
t_H	I or I/O Hold after CLK High	0			0			0			ns
t_{CO}	CLK High to Output Valid			16			17			20	ns
t_{CNT}	Register Output Feedback to Register Input — Internal Path			30			35			40	ns
t_{CH}	CLK High Time	10			11			12			ns
t_{CL}	CLK Low Time	10			11			12			ns

SWITCHING WAVEFORMS

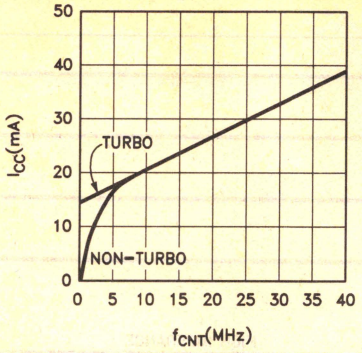
COMBINATORIAL MODE



SYNCHRONOUS CLOCK MODE



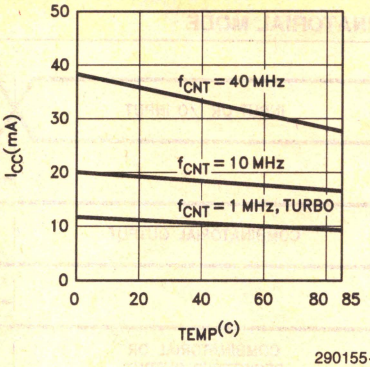
Current in Relation to Frequency



290155-11

Conditions: T_A = 0°C, V_{CC} = 5.25V

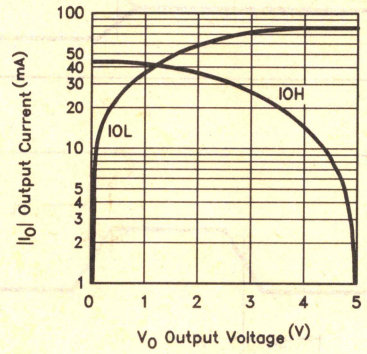
Current in Relation to Temperature



290155-12

Conditions: V_{CC} = 5.25V

Output Drive Current in Relation to Voltage



290155-13

Conditions: T_A = +25°C, V_{CC} = 5V

5C060/5C090
600-/900-GATE CHMOS H-SERIES ERASABLE PROGRAMMABLE LOGIC DEVICE (H-EPLD)

- High Performance LSI Semi-Custom Logic Replacement for Low-End Gate Arrays, TTL, and 74HC SSI and MSI Logic
- CHMOS EPROM Technology Based. UV Erasable
- Low Power; 50 μA Typical Standby Current
- Erasable Array for 100% Generic Testability
- Programmable Clock System with Two Synchronous Clocks as Well as Asynchronous Clocking Option on all Registers
- Programmable Output Registers. Can be Configured as D, T, SR, or JK Types
- Programmable Security Bit Allows Total Protection of Proprietary Designs

5C060 FEATURES:

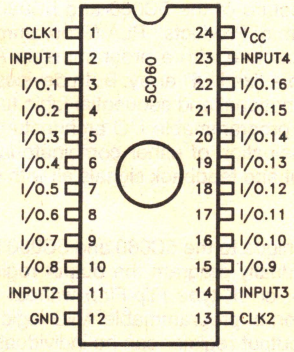
- 16 Macrocells with Programmable I/O Architecture; up to 20 Inputs (4 Dedicated, 16 I/O) or 16 Outputs
- High Speed t_{PD} (max) 45 ns, 16.67 MHz Performance
- High Performance Upgrade for Commonly Used 24-Pin PLDs
- Small Footprint 24-Pin 0.3" DIP Package
- 28 Pin J-Leaded Chip Carrier Package

(See Packaging Spec. Order #231369)

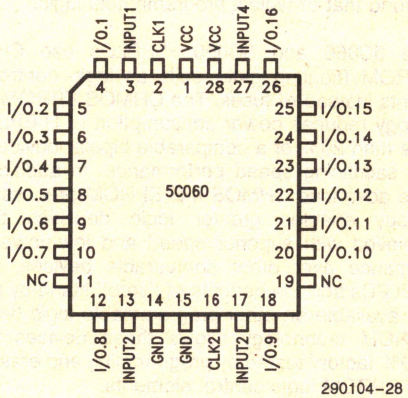
5C090 FEATURES:

- 24 Macrocells with Programmable I/O Architecture; Up to 36 Inputs (12 Dedicated, 24 I/O) or 24 Outputs
- High Speed t_{PD} (max) 50 ns, 16 MHz Performance
- Logic and I/O Superset of the 5C060
- 40-Pin DIP Package for Expanded I/O Capability
- 44-Pin J-Leaded Chip Carrier Package

(See Packaging Spec., Order Number #231369)



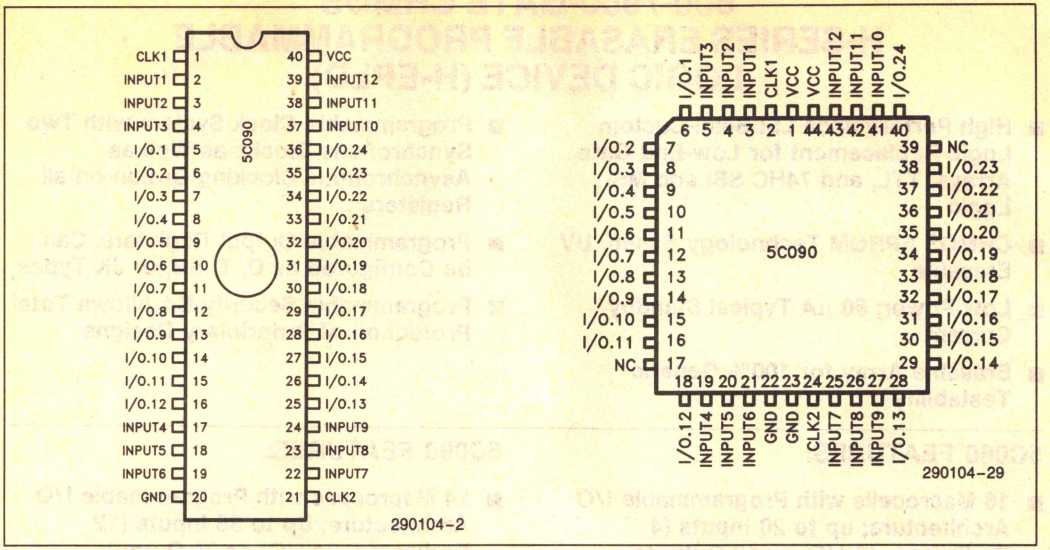
290104-1



290104-28

5C060 Pin Configurations

November 1987
Order Number: 290104-005



5C090 Pin Configurations

The Intel 5C060 and 5C090 H-EPLDs (H-series Programmable Logic Devices) are capable of implementing over 600 and 900 respectively of equivalent gates of user-customized logic functions through programming. Both devices can be used to replace low-end gate arrays, multiple programmable logic arrays and LS TTL and 74HC (CMOS) SSI and MSI logic devices. The 5C060 can also be used as a direct, low-power replacement for most, common 24-pin fuse-based programmable logic devices. With their revolutionary programmable I/O architecture, both devices have advanced functional capabilities beyond that of typical programmable logic.

The 5C060 and 5C090 H-EPLDs use CHMOS EPROM (floating gate) cells as logic control elements instead of fuses. The CHMOS EPROM technology reduces power consumption of H-EPLDs to less than 20% of a comparable bipolar device without sacrificing speed performance. In addition, Intel's advanced CHMOS II-E EPROM process technology enables greater logic densities to be achieved with superior speed and low-power performance over other comparable devices. Intel's H-ELPDs add the benefits of "zero" stand-by power not available on other programmable logic devices. EPROM technology allows these devices to be 100% factory tested by programming and erasing all the EPROM logic control elements.

The erasability of EPLDs introduces the designer to a new concept in hardware design called Modular EPLD Logic Design (MELD). Just as modular software design speeds development time and reduces errors by isolating them to a specific module, the MELD philosophy aids in hardware design. A designer can develop his modular design on the Intel Programmable Logic Development System II (iPLDS II) and test individual modules for functionality. If one of the modules has a design flaw, the designer merely erases the part and starts anew (since the 5C060 and 5C090 are EPROM-based, there is no waste associated with modular design as there would be in fuse-based PLDs).

The architecture of the 5C060 and 5C090 is based on the "Sum of Products" PLA (Programmable Logic Array) structure with a programmable AND array feeding into a fixed OR array. Both devices accomodate combinational and sequential logic functions. A proprietary programmable I/O architecture provides individual selection of either combinatorial or registered output and feedback signals all with selectable polarity.

A feature unique to the 5C060 and 5C090 is the ability to individually program the output registers as a D-, T-, SR-, or JK-type Flip-Flop without sacrificing the utilization of programmable AND logic. Additionally, each output register can be individually clocked from any of the input or feedback paths available

within the AND array. With these features, a wide variety of logic functions can be simultaneously implemented–all on the same device.

ARCHITECTURE DESCRIPTION

Externally, the 5C060 has 4 dedicated data input pins, 16 I/O pins which may be configured for input, output, or bidirectional operations, and 2 synchronous clock inputs. The 5C060 is contained in a 24-pin windowed package (0.3 inch wide), and contains 16 programmable registers.

The 5C090 represents a superset of the 5C060 in capability. The 5C090 has 12 dedicated inputs, 24 I/O pins which may be configured for input, output, or bidirectional operations, and 2 synchronous clock inputs. The 5C090 is packaged in a 40-lead windowed ceramic DIP and contains 24 programmable registers.

The basic Macrocell architecture for both the 5C060 and 5C090 is shown in Figure 1. The 5C060 has 16 of these Macrocells while the 5C090 has 24 (one for each I/O pin). The Macrocell is organized in the familiar sum-of-products structure with a programmable AND array attached to a fixed OR term. The inputs to the programmable AND array originate from the true and complement signals from each of the dedicated input pins and each of the I/O control blocks. The 40-input AND array of the 5C060 feeds 160 AND gates (product terms) which are distributed among the 16 available Macrocells within that device.

The AND array for the 5C090 has 72 inputs derived from the true and complement signals at the input and I/O pins. The AND array in the 5C090 encompasses 240 product terms which are distributed among the 24 Macrocells. The global device architectures are shown in Figure 2.

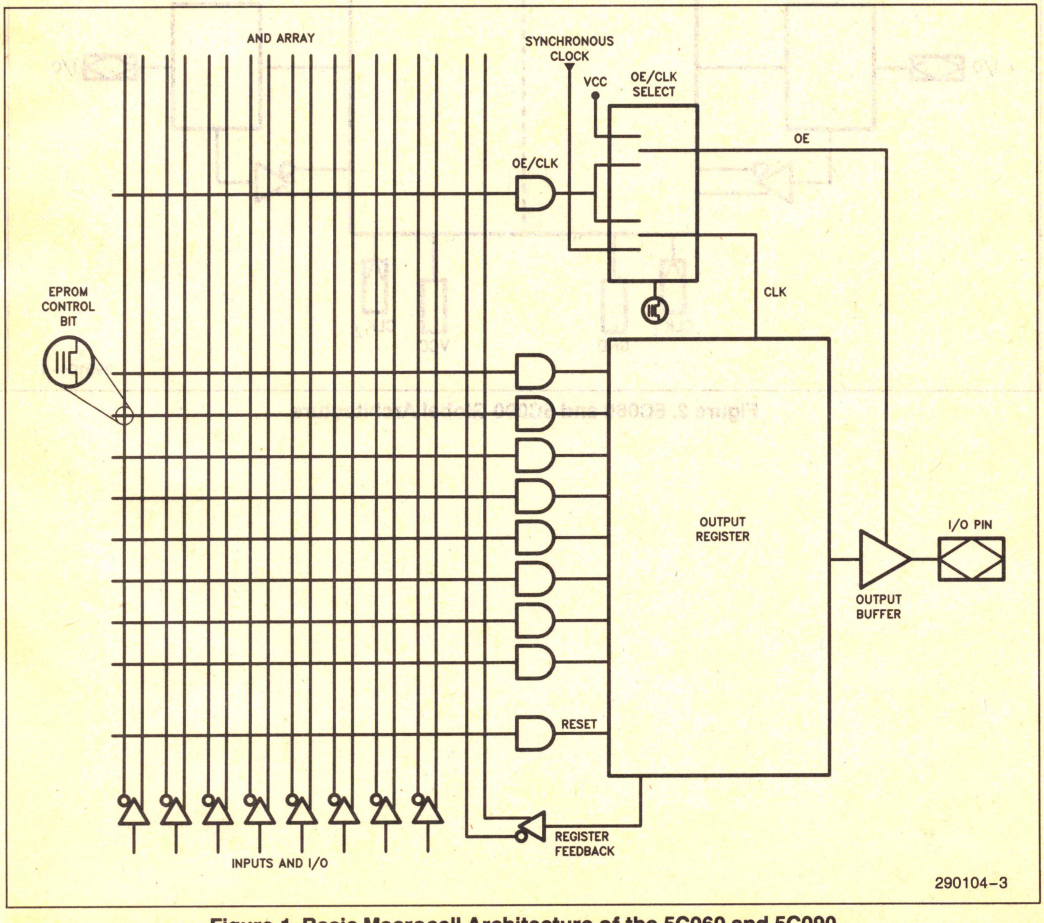


Figure 1. Basic Macrocell Architecture of the 5C060 and 5C090

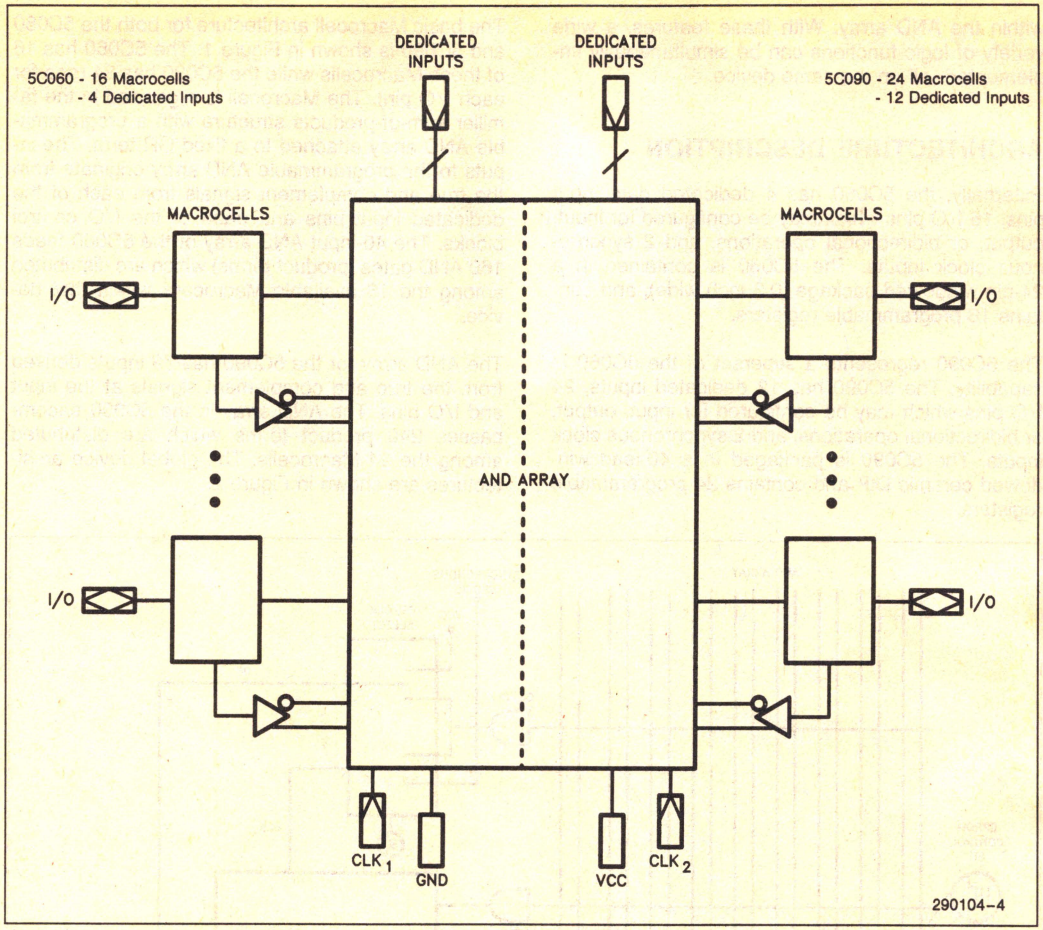


Figure 2. 5C060 and 5C090 Global Architecture

The Macrocells on both devices contain ten product terms total. Eight of the ten product terms (AND gates) are dedicated for logic implementation. One product term on each Macrocell is used for RESET control to the output register associated with the Macrocell. The final product term is used for OUTPUT ENABLE/Asynchronous Clock implementation.

Within the AND array, there is an EPROM connection at every intersection of an input signal (true and complement) and a product term to a given Macrocell. Before programming an erased device, every EPROM connection is made at every intersection. But during the programming process, these connections are opened so that only the desired connections remain. Therefore, the true or complement of any input signal can be connected to any product term. If both the true and complement connections of any signal are left intact, a logical false results on the output of the AND gate. However, if both the true and complement connections are open, then a logic "don't care" results on the AND gate. Lastly, if all the inputs of a product term are programmed open, then a logical true results on the output of the AND gate.

Both the 5C060 and 5C090 have two dedicated clock inputs to provide synchronous clock signals to the internal registers. Each of the clock signals controls half the total registers within the given device. For example, CLK1 provides synchronous clocking to the registers in Macrocells in the left half of the array while CLK2 controls the registers associated with Macrocells in the right half of the array. The advanced I/O architecture allows for any number of the registers to be synchronously clocked (from none to all). Both of the dedicated clock inputs latch the data into a given register when triggered on a positive edge.

MACROCELL ARCHITECTURE SELECTION

The 5C060 and 5C090 architecture provides each Macrocell with over 50 different possible I/O register configurations. Each I/O pin can be configured for combinatorial or registered output (true or complement) with feedback. In addition, four different types of output registers can be implemented into every I/O pin without any additional logic requirements. The feedback mechanism for each register back into the AND array can be programmed to provide for either registered feedback from the Macrocell or input feedback (treating the pin as an input). Another advantage of the advanced I/O capability of the 5C060 and the 5C090 is the ability to individually clock each internal register from asynchronous clock signals.

Output Enable (OE)/Clock Selection

Two modes of operation are provided by the OE/CLK Select Multiplexer as a part of each Macrocell. One mode provides for three-state buffering of outputs while in the other mode, the outputs are always enabled. The operation of the OE/CLK Select Multiplexer sets the mode within a given Macrocell. Therefore, the output mode can be selected individually on every output. Figure 3 illustrates the two modes of OE/CLK operation.

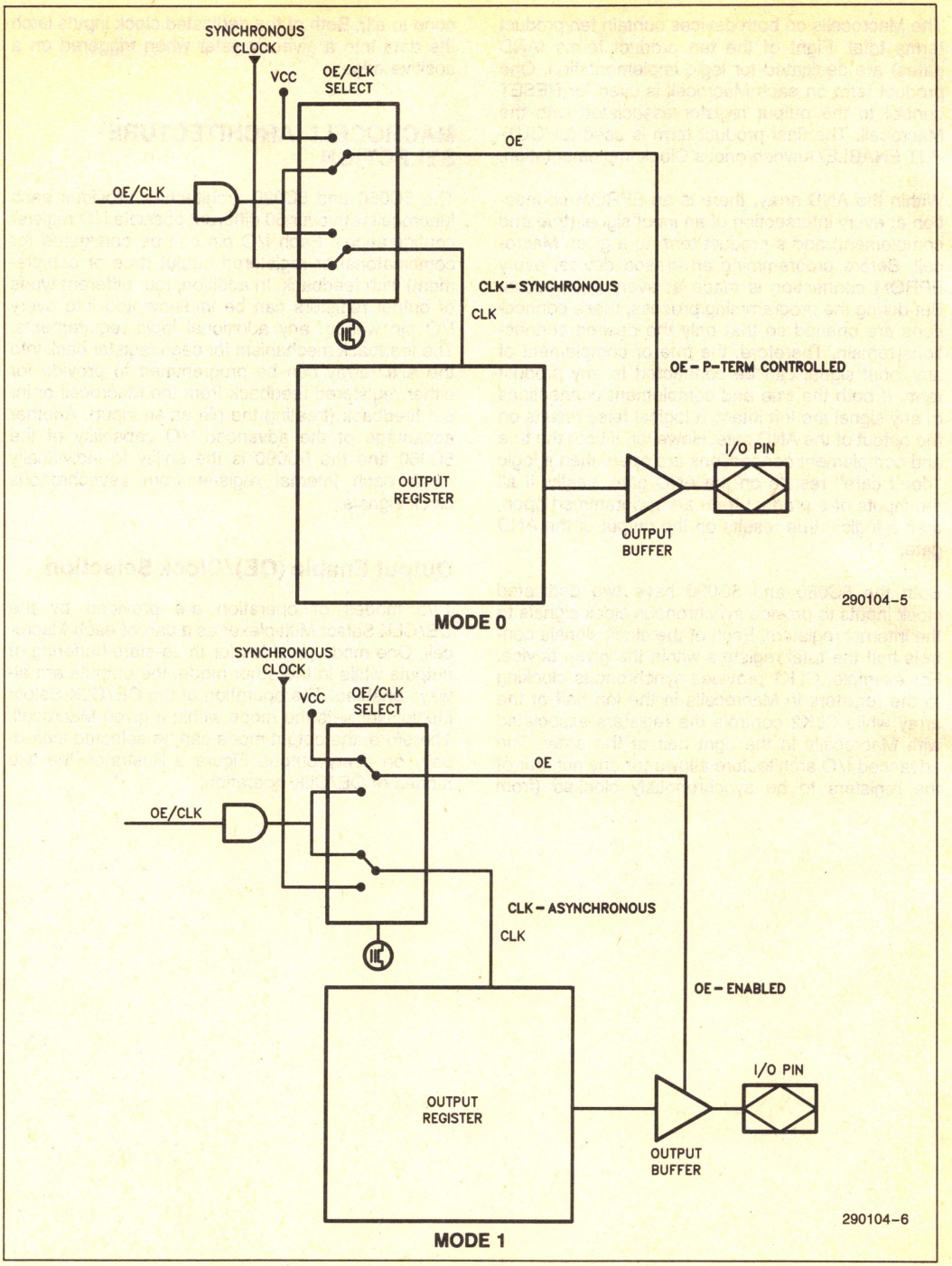


Figure 3. Output Enable/Clock Configuration

MODE 0: THREE-STATE BUFFERING

In Mode 0, the three-state output buffer is controlled by a single product term originating from the AND array. The output is enabled when the product term is a logical true. Conversely, the output appears as high impedance when the product term is a logical false as shown in Table 1. In Mode 0, the Macrocell Flip-Flop is connected to its associated synchronous clock (either CLK1 or CLK2 depending upon the Macrocell's location within the device). Thus, the Macrocell Flip-Flop may be clocked by its respective synchronous clock but its output will not become valid until the output is enabled.

Table 1. Mode 0 Output Selection

Product Term	Output Buffer
FALSE	Three-State
TRUE	Enabled

MODE 1: OUTPUT BUFFER ENABLED

In Mode 1, the Output Buffer is always enabled. In addition, the Macrocell Flip-Flop is connected to the AND array. The Macrocell Flip-Flop may now be triggered from an asynchronous clock signal generated by the AND array logic to the OE/CLK multiplexable term. Mode 1 allows the Macrocell Flip-Flops to be individually clocked from any of the available signals in the AND array. Since both true and complement values appear in the AND array, the Flip-Flop may be configured to trigger on positive or negative clock edges. Gated clock structures can be created since the Flip-Flop clock is created by a product term.

Invert Select EPROM Bit

The Invert Select EPROM bit is used to invert the product term input into the register. This applies to all inputs including double inputs on the JK and SR registers.

REGISTER SELECTION

The advanced I/O architecture of the 5C060 and the 5C090 allows four different register types along with combinatorial output as illustrated in Figure 4. The register types include a T, D, JK, or SR Flip-Flop and each Macrocell I/O structure may be independently configured. In addition, all registers have an individual asynchronous RESET control from a dedicated product term derived in the AND array. When this dedicated product term is a logical one, the Macrocell register is immediately cleared to a logical zero independent of the register clock. The RESET function occurs automatically on power-up.

Output Register Configuration

The four different register types shown in Figure 4 are described below.

D- or T-type Flip-Flops

When either a D- or T-type Flip-Flop is configured as part of the I/O structure, all eight of the product terms into the Macrocell are ORed together and fed into the register input.

JK or SR Registers

When either a JK or SR register is configured, the eight product terms are shared among two OR gates (one for the J or S input and the other for the K or R input). The allocation for these product terms for each of the register inputs is optimized by the iPLDS II development software.

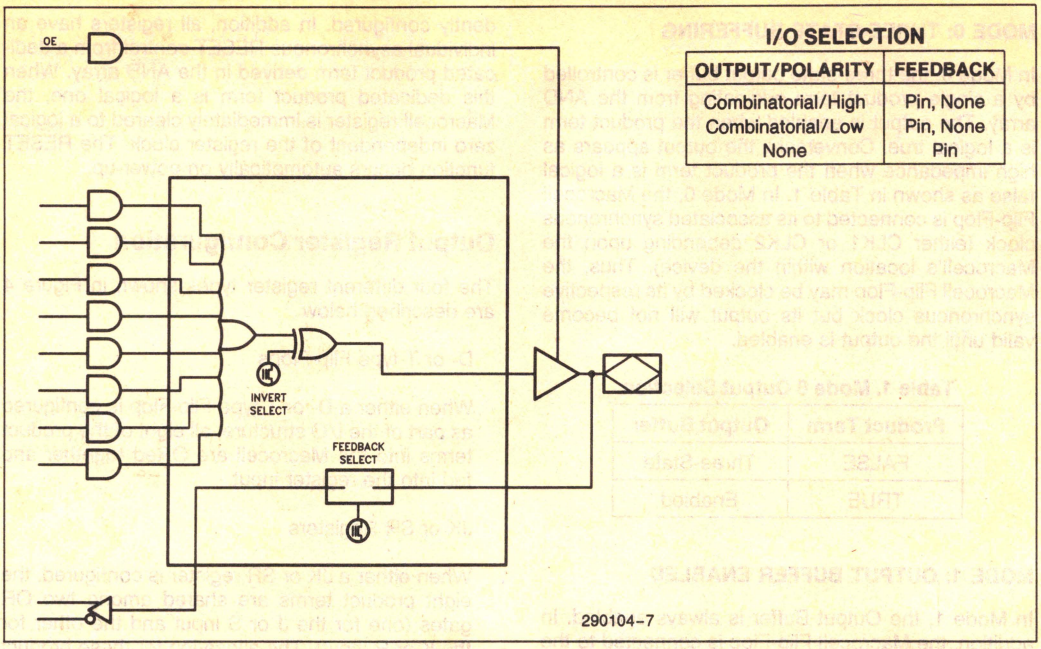
OUTPUT/FEEDBACK

The Output Select Multiplexer allows for either registered, combinatorial or no output.

The Feedback Select Multiplexer EPROM bit enables registered, I/O (using the pin for bidirectional input or just input), or no feedback to the AND array.

The Feedback Select is also important for building product terms with more than 8 products. The 8-product product term of a Macrocell can be fed back into the AND array and combined with still more signals to create a much larger product term (of more than 8-inputs). In addition, if the feedback product term is not to be output, then the iPLDS II will reserve the associated Macrocell pin and indicate it in the REPORT file. A reserved pin should be left floating (no connect) when assembled onto a circuit board.

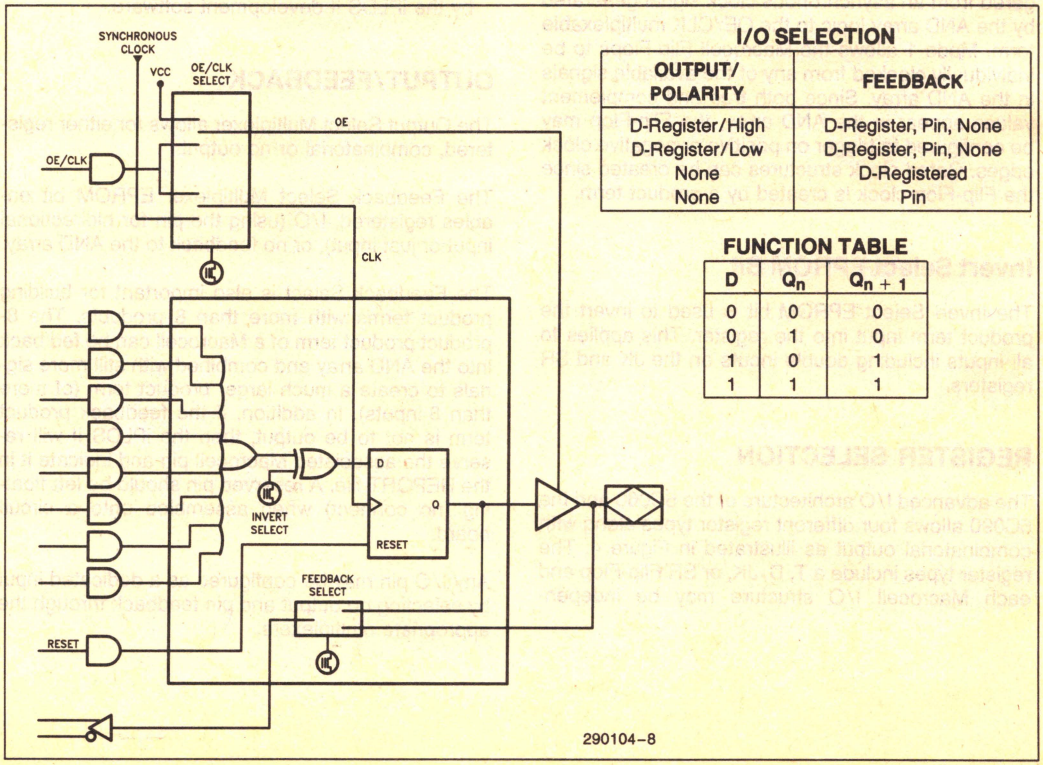
Any I/O pin may be configured as a dedicated input by selecting no output and pin feedback through the appropriate multiplexers.



I/O SELECTION

OUTPUT/POLARITY	FEEDBACK
Combinatorial/High	Pin, None
Combinatorial/Low	Pin, None
None	Pin

Figure 4a. Combinatorial I/O Configuration



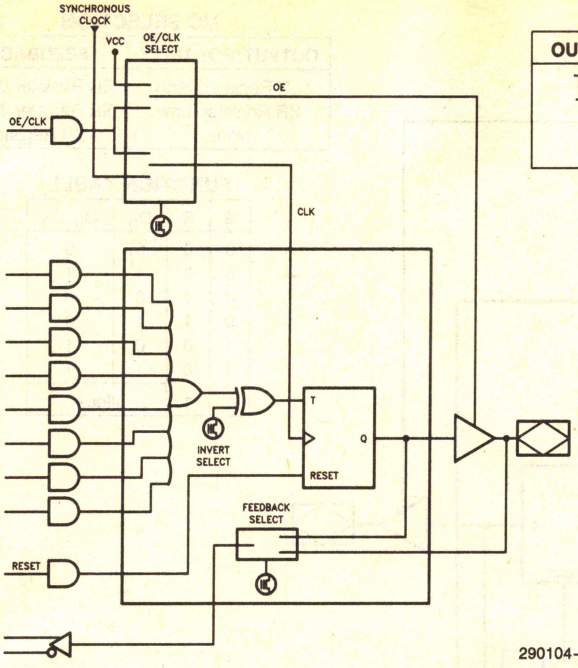
I/O SELECTION

OUTPUT/ POLARITY	FEEDBACK
D-Register/High	D-Register, Pin, None
D-Register/Low	D-Register, Pin, None
None	D-Registered
None	Pin

FUNCTION TABLE

D	Q_n	Q_{n+1}
0	0	0
0	1	0
1	0	1
1	1	1

Figure 4b. D-Type Flip-Flop Register Configuration



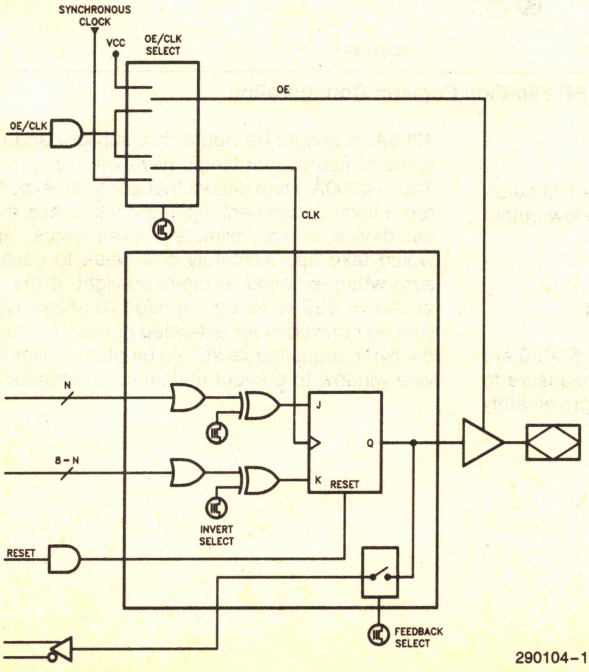
I/O SELECTION

OUTPUT/POLARITY	FEEDBACK
T-Register/High	T-Register, Pin, None
T-Register/Low	T-Register, Pin, None
None	T-Register
None	Pin

FUNCTION TABLE

T	Q_n	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

Figure 4c. Toggle Flip-Flop Register Configuration



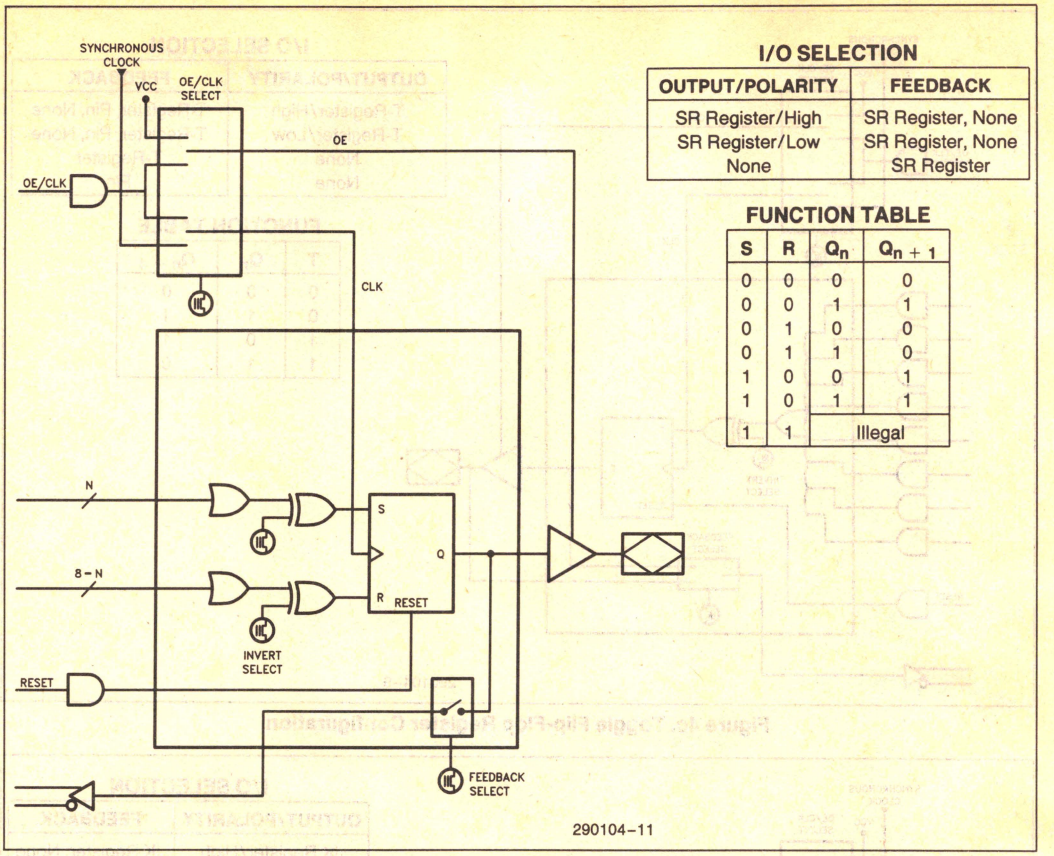
I/O SELECTION

OUTPUT/POLARITY	FEEDBACK
JK Register/High	JK Register, None
JK Register/Low	JK Register, None
None	JK Register

FUNCTION TABLE

J	K	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Figure 4d. JK Flip-Flop Register Configuration



I/O SELECTION

OUTPUT/POLARITY	FEEDBACK
SR Register/High	SR Register, None
SR Register/Low	SR Register, None
None	SR Register

FUNCTION TABLE

S	R	Q_n	$Q_n + 1$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	Illegal	

Figure 4e. SR Flip-Flop Register Configuration

Erased-State Configuration

Prior to programming or after erasing, the I/O structure is configured for combinatorial active low output with input (pin) feedback.

ERASURE CHARACTERISTICS

Erasure characteristics of the 5C060 and 5C090 are such that erasure begins to occur upon exposure to light with wavelengths shorter than approximately 4000Å. It should be noted that sunlight and certain types of flourescent lamps have wavelengths in the 3000–4000Å. Data shows that constant exposure to room level flourescent lighting could erase the typical device in approximately three years, while it would take approximately one week to cause erasure when exposed to direct sunlight. If the 5C060 or the 5C090 is to be exposed to these types of lighting conditions for extended periods of time, conductive opaque labels should be placed over the device window to prevent unintentional erasure.

The recommended erasure procedure for the 5C060 and 5C090 is exposure to shortwave ultraviolet light with a wavelength of 2537Å. The integrated dose (i.e., UV intensity × exposure time) for erasure should be a minimum of fifteen (15) Wsec/cm^2. The erasure time with this dosage is approximately 15 to 20 minutes using an ultraviolet lamp with a 12,000 μW/cm^2 power rating. The 5C060 or 5C090 should be placed within one inch of the lamp tubes during erasure. The maximum integrated dose the 5C060 or 5C090 can be exposed to without damage is 7258 Wsec/cm^2 (1 week at 12,000 μW/cm^2). Exposure to high intensity UV light for longer periods may cause permanent damage to the device.

PROGRAMMING CHARACTERISTICS

Initially, and after erasure, all the EPROM control bits of the 5C060 and 5C090 are connected (in the "1" state). Each of the connected control bits are selectively disconnected by programming the EPROM cells into their "0" state. Programming voltage and waveform specifications are available by request from Intel to support programming of the 5C060 and 5C090.

int$_e$ligent Programming™ Algorithm

Both the 5C060 and 5C090 support the int$_e$ligent Programming Algorithm which rapidly programs Intel H-ELPDs (and EPROMs) using an efficient and reliable method. The int$_e$ligent Programming Algorithm is particularly suited to the production programming environment. This method greatly decreases the overall programming time while programming reliability is ensured as the incremental program margin of each bit is continually monitored to determine when the bit has been successfully programmed.

FUNCTIONAL TESTING

Since the logical operation of the 5C060 and 5C090 are controlled by EPROM elements, the device is completely testable. Each programmable EPROM bit controlling the internal logic is tested using application-independent test program patterns. After testing, the devices are erased before shipment to customers. No post-programming tests of the EPROM array are required.

The testability and reliability of EPROM-based programmable logic devices is an important feature over similar devices based on fuse technology. Fuse-based programmable logic devices require a user to perform post-programming tests to insure proper programming. These tests must be done at the device level because of the cummulative error effect. For example, a board containing ten devices each possessing a 2% device fallout translates into an 18% fallout at the board level (it should be noted that programming fallout of fuse-based programmable logic devices is typically 2% or higher).

To enable functional evaluation of counter and state-machine applications, the 5C060 and 5C090 contain register pre-load circuitry. This can be activated by interrupting the normal clocked sequence and applying V_{pp} on pin 11 for the 5C060 or pin 17 for the 5C090 to engage the pre-load state. Under these conditions, the Flip-Flops in the 5C060 and 5C090 can be set to any logical condition and then return to normal operation. This process simplifies the input sequences necessary to evaluate the counter and state machine operations.

DESIGN RECOMMENDATIONS

To take maximum advantage of EPLD technology, it is recommended that the designer use the Modular EPLD Logic Design (MELD) method. The MELD philosophy is derived from the modular programming method used in software development. In a modular software development environment, the engineer designs a modular program (typically on a development system), stores it in memory (EPROM), and tests the module for functionality. A hardware designer using EPLDs can use this same approach when designing logic. The designer develops a modular logic design on the Intel Programmable Logic Development System II (iPLDS II), stores it in "memory" (the EPROM control elements of the EPLD), and again tests the module for functionality. If the design is in error, the logic designer reprograms the EPLD with his new design as easily as a software designer can download a new program into memory.

The MELD philosophy is new to programmable logic because EPROM-based PLDs are new. A modular logic development process using fused-based PLDs would be wasteful since a fused-based device cannot be erased an re-used.

For proper operation, it is recommended that all input and output pins be constrained to the voltage range GND < (V_{IN} or V_{OUT}) < V_{CC}. Unused inputs should be tied to an appropriate logic level (e.g. either V_{CC} or GND) to minimize device power consumption. Reserved pins (as indicated in the iPLDS II REPORT file) should be left floating (no connect) so that the pin can attain the appropriate logic level. A power supply decoupling capacitor of at least 0.2 μF must be connected directly between V_{CC} and GND pins of the 5C060 and the 5C090.

DESIGN SECURITY

A single EPROM bit provides a programmable design security feature that controls the access to the data programmed into the device. If this bit is set, a proprietary design within the device cannot be copied. This EPROM security bit enables a higher degree of design security than fused-based devices since programmed data within EPROM cells is invisible even to microscopic evaluation. The EPROM security bit, along with all the other EPROM control bits, will be reset by erasing the device.

LATCH-UP IMMUNITY

All of the input, I/O, and clock pins of the 5C060 and 5C090 have been designed to resist latch-up which is inherent in inferior CMOS structures. The 5C060 and 5C090 are designed with Intel's proprietary CHMOS II-E EPROM process. Thus, each of the 5C060 and 5C090 pins will not experience latch-up with currents up to 100 mA and voltages ranging from -1V to $V_{CC} + 1$V. Furthermore, the programming pin is designed to resist latch-up to the 13.5V maximum device limit.

INTEL PROGRAMMABLE LOGIC DEVELOPMENT SYSTEM II (iPLDS II)

The iPLDS II graphically shown in Figure 5 provides all the tools needed to design with Intel H-Series EPLDs or compatible devices. In addition to providing development assistance, iPLDS II insulates the user from having to know all the intricate details of EPLD architecture (the machine will optimize a design to benefit from architectual features). It contains comprehensive third generation software that supports four different design entry methods, minimizes logic, does automatic pin assignments and produces the best design fit for the selected EPLD. It is user friendly with guided menus, on-line Help messages and soft key inputs.

In addition, the iPLDS II contains programmer hardware in the form of an iUP-PC Universal Programmer Personal Computer to enable the user to program EPLDs, read and verify programmed devices and also to graphically edit programming files. The software generates industry standard JEDEC object code output files which can be downloaded to other programmers as well.

The iPLDS II has interfaces to popular schematic capture packages (including Dash series from FutureNet* and PC CAPS from PCAD)** to enable designs to be entered using schematics. A more integrated schematic entry method is provided by SCHEMA II-PLD, a low-cost schematic capture package that supports EPLD primitives and user-defined macro symbols. SCHEMA II-PLD contains the EPLD Design Manager, which provides a single user interface to both SCHEMA II-PLD and iPLS II software.The other design formats supported are Boolean equation entry and State Machine design entry.

The iPLDS II operates on the IBM† PC/XT, PC/AT, or other compatible machine with the following configuration:

1. At least one floppy disk drive and hard disk drive.
2. MS-DOS†† Operating System Version 3.0 or greater.
3. 640K Memory.
4. Intel iUP-PC Universal Programmer Personal Computer and GUPI Adaptor (supplied with iPLDS II).
5. A color monitor is suggested.

Detailed information on the Intel Programmable Logic Development System II is contained in a separate Intel data sheet. (Order Number: 280168)

*FutureNet is a registered trademark of FutureNet Corporation. DASH is a trademark of FutureNet Corporation.

**PC-CAPS is a trademark of P-CAD Corporation.

†IBM Personal Computer is a registered trademark of International Business Machines Corporation.

††MS-DOS is a registered trademark of Microsoft Corporation.

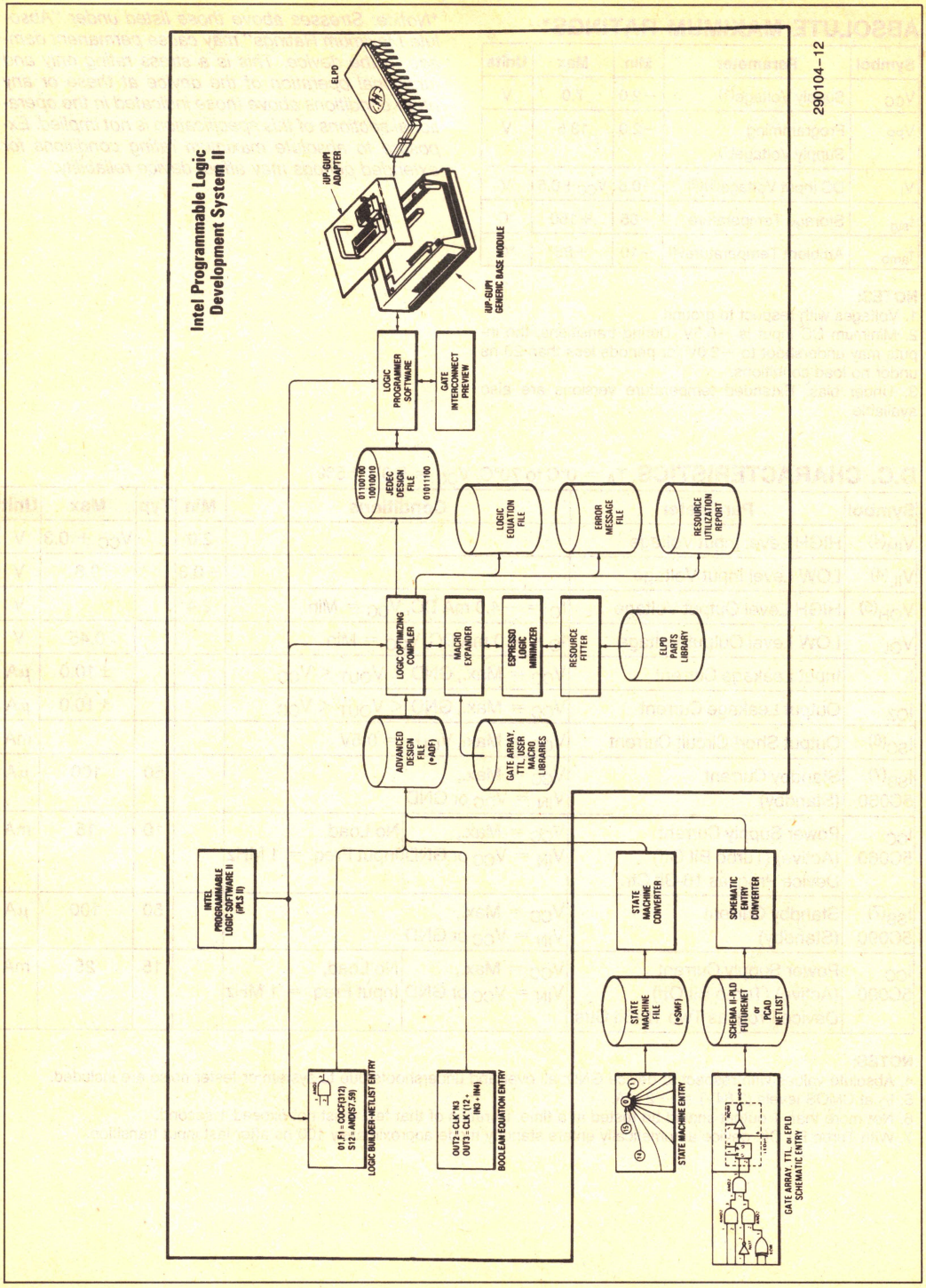


Figure 5. Intel Programmable Logic Development System II (iPLDS II)

ABSOLUTE MAXIMUM RATINGS*

Symbol	Parameter	Min	Max	Units
V_{CC}	Supply Voltage(1)	−2.0	7.0	V
V_{PP}	Programming Supply Voltage(1)	−2.0	13.5	V
V_I	DC Input Voltage(1)(2)	−0.5	V_{CC}+0.5	V
t_{stg}	Storage Temperature	−65	+150	°C
t_{amb}	Ambient Temperature(3)	−10	+85	°C

Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

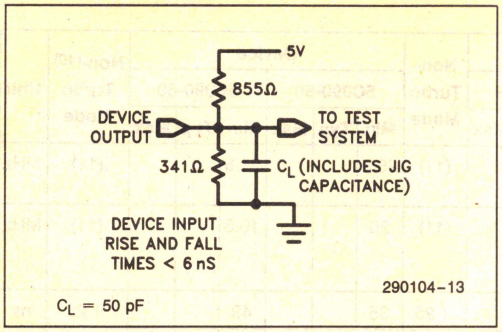
NOTES:
1. Voltages with respect to ground.
2. Minimum DC input is −0.5V. During transitions, the inputs may undershoot to −2.0V for periods less than 20 ns under no load conditions.
3. Under bias. Extended temperature versions are also available.

D.C. CHARACTERISTICS T_A = 0°C to 70°C, V_{CC} = 5.0V ±5%

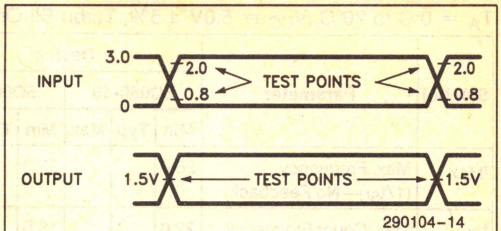
Symbol	Parameter	Conditions		Min	Typ	Max	Unit
V_{IH}(4)	HIGH Level Input Voltage			2.0		V_{CC} + 0.3	V
V_{IL}(4)	LOW Level Input Voltage			−0.3		0.8	V
V_{OH}(5)	HIGH Level Output Voltage	I_O = −4.0 mA DC, V_{CC} = Min.		2.4			V
V_{OL}	LOW Level Output Voltage	I_O = 4.0 mA DC, V_{CC} = Min.				0.45	V
I_I	Input Leakage Current	V_{CC} = Max., GND < V_{OUT} < V_{CC}				±10.0	µA
I_{OZ}	Output Leakage Current	V_{CC} = Max., GND < V_{OUT} < V_{CC}				±10.0	µA
I_{SC}(6)	Output Short Circuit Current	V_{CC} = Max., V_{OUT} = 0.5V					mA
I_{SB}(7) 5C060	Standby Current (Standby)	V_{CC} = Max., V_{IN} = V_{CC} or GND			50	100	µA
I_{CC} 5C060	Power Supply Current (Active) (Turbo Bit Off) Device Prog. as 16-Bit Ctr.	V_{CC} = Max., V_{IN} = V_{CC} or GND	No Load, Input Freq. = 1 MHz		10	15	mA
I_{SB}(7) 5C090	Standby Current (Standby)	V_{CC} = Max., V_{IN} = V_{CC} or GND			50	100	µA
I_{CC} 5C090	Power Supply Current (Active) (Turbo Bit Off) Device Prog. as Two 12-Bit Ctrs.	V_{CC} = Max., V_{IN} = V_{CC} or GND	No Load, Input Freq. = 1 MHz		15	25	mA

NOTES:
4. Absolute values with respect to device GND; all over and undershoots due to system or tester noise are included.
5. I_O at CMOS levels (3.84V) = −2 mA.
6. Not more than 1 output should be tested at a time. Duration of that test must not exceed 1 second.
7. With Turbo Bit Off, device automatically enters standby mode approximately 100 ns after last input transition.

A.C. TESTING LOAD CIRCUIT



A.C. TESTING INPUT, OUTPUT WAVEFORM



A.C. Testing: Inputs are Driven at 3.0V for a Logic "1" and 0V for a Logic "0". Timing Measurements are made at 2.0V for a Logic "1" and 0.8V for a Logic "0" on inputs. Outputs are measured at a 1.5V point.

A.C. CHARACTERISTICS T_A = 0°C to 70°C, V_{CC} = 5V ±5%, Turbo Bit On[8]

Symbol	From	To	Device												Non-[10] Turbo Mode	Unit
			5C060-45			5C060-55			5C090-50			5C090-60				
			Min	Typ	Max	Min	Typ	Max	Min	Typ	Max	Min	Typ	Max		
t_{PD1}	Input	Comb. Output			43			53			46			55	+25	ns
t_{PD2}	I/O	Comb. Output			45			55			50			60	+25	ns
t_{PZX}[9]	I or I/O	Output Enable			45			55			50			60	+25	ns
t_{PXZ}[9]	I or I/O	Output Disable			45			55			50			60	+25	ns
t_{CLR}	Asynch. Reset	Q Reset			45			55			50			60	+25	ns

NOTES:
8. Typical Values are at T_A = 25°C, V_{CC} = 5V, Active Mode.
9. t_{PZX} and t_{PXZ} are measured at ±0.5V from steady state voltage as driven by spec. output load. t_{PXZ} is measured with C_L = 5 pF.
10. If device is operated with Turbo Bit Off (Non-Turbo Mode), increase time by amount shown.

CAPACITANCE

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
C_{IN}	Input Capacitance	V_{IN} = 0V, f = 1.0 MHz			20	pF
C_{OUT}	Output Capacitance	V_{OUT} = 0V, f = 1.0 MHz			20	pF
C_{CLK}	Clock Pin Capacitance	V_{OUT} = 0V, f = 1.0 MHz			20	pF
C_{VPP}	V_{PP} Pin	Pin 13 on 5C060			50	pF
		Pin 21 on 5C090			80	pF

SYNCHRONOUS CLOCK MODE A.C. CHARACTERISTIC

T_A = 0°C to 70°C, V_{CC} = 5.0V ±5%, Turbo Bit On(8)

Symbol	Parameter	Device						Non-Turbo Mode	Device						Non-(10) Turbo Mode	Unit
		5C060-45			5C060-55				5C090-50			5C090-60				
		Min	Typ	Max	Min	Typ	Max		Min	Typ	Max	Min	Typ	Max		
f_{MAX}	Max. Frequency ($1/t_{SU}$—No Feedback)	26.0			23.0			(11)	26.0			21.5			(11)	MHz
f_{CNT}	Max. Count Frequency ($1/t_{CNT}$—With Feedback)	22.0			18.0			(11)	20			16.5			(11)	MHz
t_{SU1}	Input Setup Time to CLK	36			41			+25	36			43			+25	ns
t_{SU2}	I/O Setup Time to CLK	38			43			+25	38			46			+25	ns
t_H	I or I/O Hold after CLK High	0			0				0			0				ns
t_{CO}	CLK High to Output Valid			22			25				23			25		ns
t_{CNT}	Register Output Feedback to Register Input—Internal Path			45			55	+25			50			60	+25	ns
t_{CH}	CLK High Time	17.5			21.5				17.5			23				ns
t_{CL}	CLK Low Time	17.5			21.5				17.5			23				ns

ASYNCHRONOUS CLOCK MODE A.C. CHARACTERISTICS

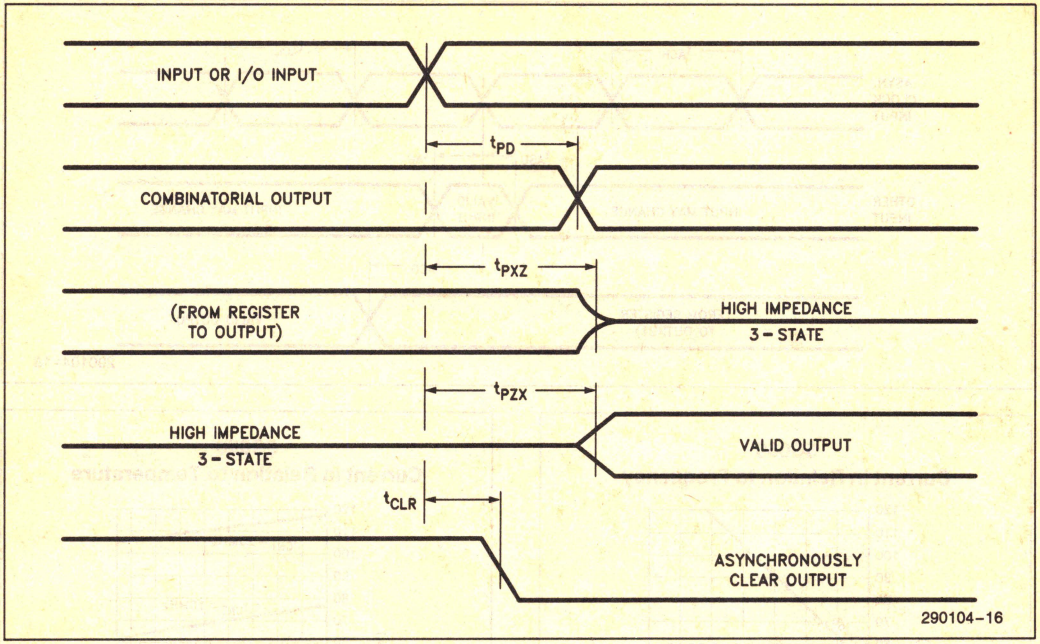
T_A = 0°C to 70°C, V_{CC} = 5.0V ±5%, Turbo Bit On(8)

Symbol	Parameter	Device						Non-Turbo Mode	Device						Non-(10) Turbo Mode	Unit
		5C060-45			5C060-55				5C090-50			5C090-60				
		Min	Typ	Max	Min	Typ	Max		Min	Typ	Max	Min	Typ	Max		
f_{ACNT}	Max. Count Frequency ($1/t_{ACNT}$—With Feedback)	22.0			18.0			(11)	20			16.5			(11)	MHz
t_{ASU1}	Input Setup Time to Asynch. Clock	10			10			+25	10			10			+25	ns
t_{ASU2}	I/O Setup Time to Asynch. Clock	12			12			+25	10			10			+25	ns
t_{AH}	Input or I/O Hold After Asynch. Clock	15			15				15			15				ns
t_{ACO}	Asynch. CLK to Output Valid			52			62				60			70		ns
t_{ACNT}	Register Output Feedback to Register Input—Internal Path			45			55	+25			50			60	+25	ns
t_{ACH}	Asynch. CLK High Time	17.5			21.5				20			25				ns
t_{ACL}	Asynch. CLK Low Time	17.5			21.5				20			25				ns

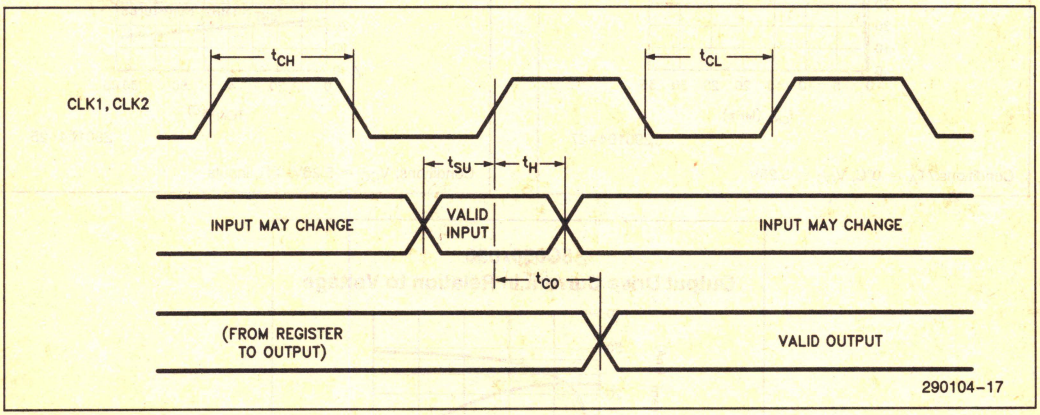
NOTES:
11. Recalculate frequency according to equation at left of table.

SWITCHING WAVEFORMS

COMBINATORIAL MODE

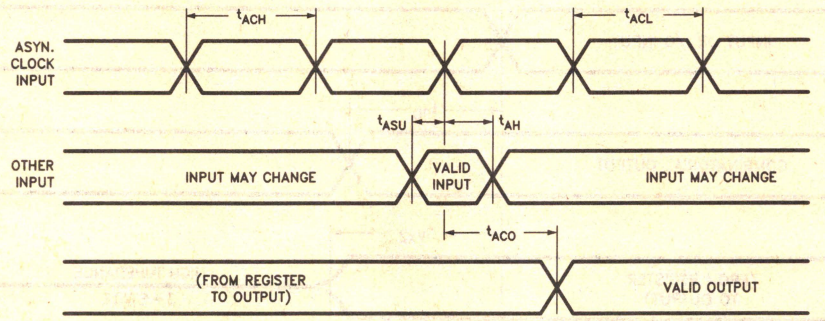


SYNCHRONOUS CLOCK MODE



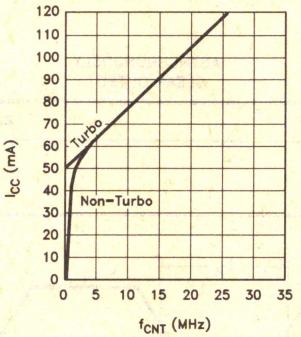
SWITCHING WAVEFORMS (Continued)

ASYNCHRONOUS CLOCK MODE



290104-18

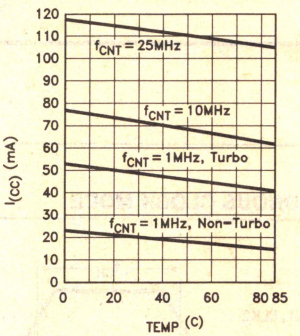
5C060
Current in Relation to Frequency



290104-27

Conditions: T_A = 0°C, V_{CC} = 5.25V

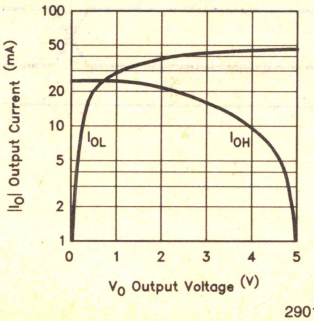
5C060
Current in Relation to Temperature



290104-25

Conditions: V_{CC} = 5.25V, TTL inputs

5C060/090
Output Drive Current in Relation to Voltage



290104-26

Conditions: T_A = 25°C

5C121
1200 GATE CHMOS
H-SERIES ERASABLE PROGRAMMABLE LOGIC DEVICE

- **High Performance LSI Semi-Custom Logic Replacement for Gate Arrays and Conventional Fixed Logic**
- **EPROM Technology Based. UV Erasable**
- **Programmable Macrocell and I/O Architecture; up to 36 Inputs or 24 Outputs, 28 Macrocells Including 4 Buried Registers**
- **All Inputs are Latchable with a Programmable Latch Feature**
- **High Speed t_{PD} (Max) 50 ns Operating Frequency (Max) 20 MHz**
- **Low Power; 15 mW Typical Standby Dissipation**
- **Typical Usable Gate Count of 1200 2-Input NAND Gates**
- **Advanced Architecture Features Including Programmable Output Polarity (Active High/Low), Register By-Pass and Reset Controls**
- **Programmable Clock System for Input Latches and Output Registers**
- **Product-Term Sharing and Local Bus Architecture for Optimized Array Performance**
- **Compatible with LS TTL and 74HC CMOS Logic**
- **Register Pre-Load and Erasable Array for 100% Generic Testability**
- **Programmable "Security Bit" allows total protection of proprietary designs**
- **Available in a 40-Lead Window Cerdip Package** (See Packaging Spec, Order #231369)

The Intel 5C121 H-EPLD (H-series Erasable Programmable Logic Device) is an LSI logic circuit that is user customizable through programming. This device can be used to replace gate arrays, multiple programmable logic arrays and LS TTL and 74HC CMOS SSI and MSI logic devices. The logic capacity of the 5C121 is typically equal to 1200 two-input NAND gates.

The 5C121 H-EPLD uses CHMOS* EPROM (floating gate) cells as logic control elements instead of fuses. Use of Intel's advanced CHMOS II-E EPROM process technology enables greater logic densities to be achieved with superior speed and power performance. The EPROM technology also enables these devices to be 100% factory tested by the programming and the erasure of all the EPROM logic control elements in the device.

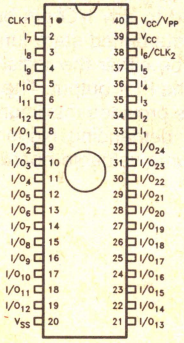
The architecture of the 5C121 is based on the 'Sum of Products' PLA (Programmable Logic Array) structure with a programmable AND array feeding into a fixed OR array. Flexibility in accommodating logical functions without the overhead of unnecessary product terms or speed penalties of programmable OR structures is achieved through the provision of a range of OR gate widths as well as through product term sharing. The use of a segmented PLA structure with local and global connectivity allows for further improvements in performance. The 5C121 also contains innovative architectural features that provide extensive Input/Output flexibility.

*CHMOS is a patented process of Intel Corporation.

RECOMMENDED OPERATING CONDITIONS

Symbol	Parameter	Min	Max	Units
V_{CC}	Supply Voltage	4.75	5.25	V
V_I	INPUT Voltage	0	V_{CC}	V
V_O	OUTPUT Voltage	0	V_{CC}	V
T_A	Operating Temperature	0	70	°C
t_R	INPUT rise Time		500	ns
t_F	INPUT fall Time		500	ns

Pin Configuration



290098-1

ILLUSTRATIONS COURTESY OF ALTERA CORPORATION.

November 1987
Order Number: 290098-004

ARCHITECTURE DESCRIPTION

The 5C121 H-EPLD has 12 dedicated inputs as well as 24 Input/Output pins. All inputs to the circuit (both dedicated and I/O inputs) may be latched using transparent 7475 type latches. In addition to these 36 input latches, 28 D type registers are also provided.

The internal architecture of the 5C121 H-EPLD is based on 28 macrocells. Each macrocell (see Figure 1) contains a PLA structure (programmable AND array product terms connected to an OR gate) and an I/O architecture control block (with a D Flip-Flop) that can be programmed to create many different output logic structures. This powerful I/O architecture can be configured to support both active-high, active-low, 3-state, open drain and bi-directional data ports all on a 4-bit wide basis. They can also act as inputs on a nibble wide basis with optional input latching.

Macrocells in each half of the circuit are grouped together for I/O architecture programming. Each bank of four macrocells can be further programmed on an individual macrocell basis to generate active high or active low outputs of the logic function from the PLA.

The primary logic array of the 5C121 is segmented into two symmetrical halves that communicate via global bus signals. The main array contains some 15104 programmable elements representing 236 product terms (AND gates) each containing 64 input signals.

The macrocells share a common programmable clock system (described in a later section) that controls clocking of all registers and input latches. The device contains 8 modes of clock operation that allow logic transition to take place on either rising or falling edges of the clock signals.

The device also contains four macrocells whose outputs are not tied to any I/O pin but feed back into the array to create buried state-functions. The feedback path may be either the registered or combinational result of the PLA output. The use of the buried state macrocells provides maximum equivalent logic density without demanding higher pin-count packages that consume valuable board space.

MACROCELL I/O ARCHITECTURE

The Input/Output architecture of the 5C121 macrocell (see Figure 1) can be programmed using both static and dynamic controls. The static controls remain fixed after the device is programmed whereas the dynamic controls may change state as a result of the signals applied to the device.

The static controls set the inversion logic (i), register by-pass (ii) and input feedback multiplexers (iii). In the latter two cases these controls operate on four macrocells as a bank.

The buried-state registers have simpler controls that determine if the feedback is to be registered or combinational.

The inversion control logic, marked (i) in Figure 1, is achieved by programming the EPROM control bit connected to the same XOR gate as the output from the PLA structure. Programming or erasure of this EPROM element toggles the OR gate output of the PLA between active-high and active-low. The inversion control operates on an individual macrocell basis.

The register by-pass control, marked (ii) in Figure 1 allows the PLA output to either flow through the D Flip-Flop as a registered output or by-pass the Flip-Flop and be a combinational output.

The dynamic controls consist of a programmable input latch-enable as well as reset and output enable product terms. The latch-enable function is common throughout the 5C121 and once chosen, will latch all the inputs. This function is programmed by the clock control block but may also be driven by input signals applied to pin 1 (see clock modes—Table 1).

The reset and output-enable controls are logically controlled by single product terms (the logic AND of programmed variables in the array). These terms have control over banks of four macrocells.

The output-enable control may be used to generate architecture types that include bi-directional, 3-state, open drain, or input only structures.

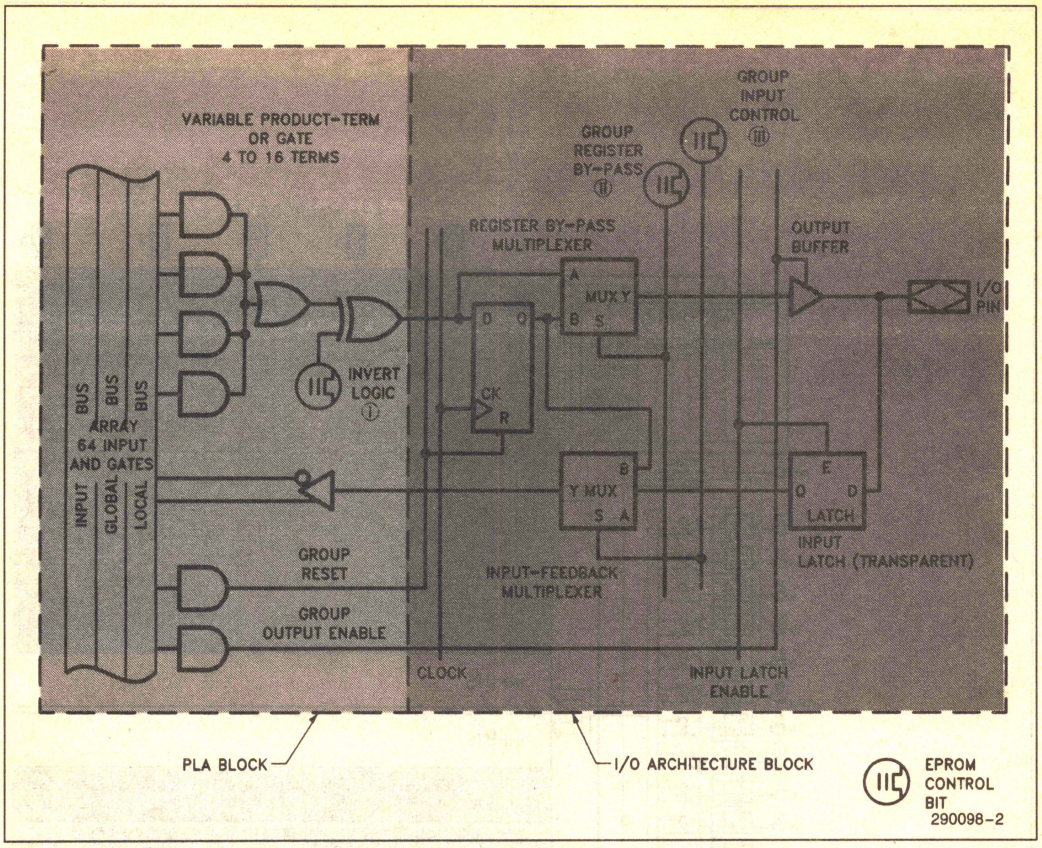


Figure 1. 5C121 Macrocell I/O Architecture

INTERNAL BUS STRUCTURE

The two identical halves of the 5C121 communicate via a series of busses. The local bus structure used for communication within each half of the chip contains 16 conductors that carry the TRUE and COMPLEMENT of 8 local macrocells. In the block diagram (Figure 2) of the 5C121 the local macrocells are B-1 and B-2 on one half and A-1 and A-2 on the other half.

The global busses (Input bus & Global feedback from A-3 & B-3 macrocells & buried registers) are made up of 48 conductors that span the entire chip. These 48 conductors carry the TRUE and COMPLEMENT of the twelve primary inputs (pins 2 through 7 and 33 through 38), signals from 4 Buried Registers as well as the global outputs of 8 macrocells in groups A-3 and B-3.

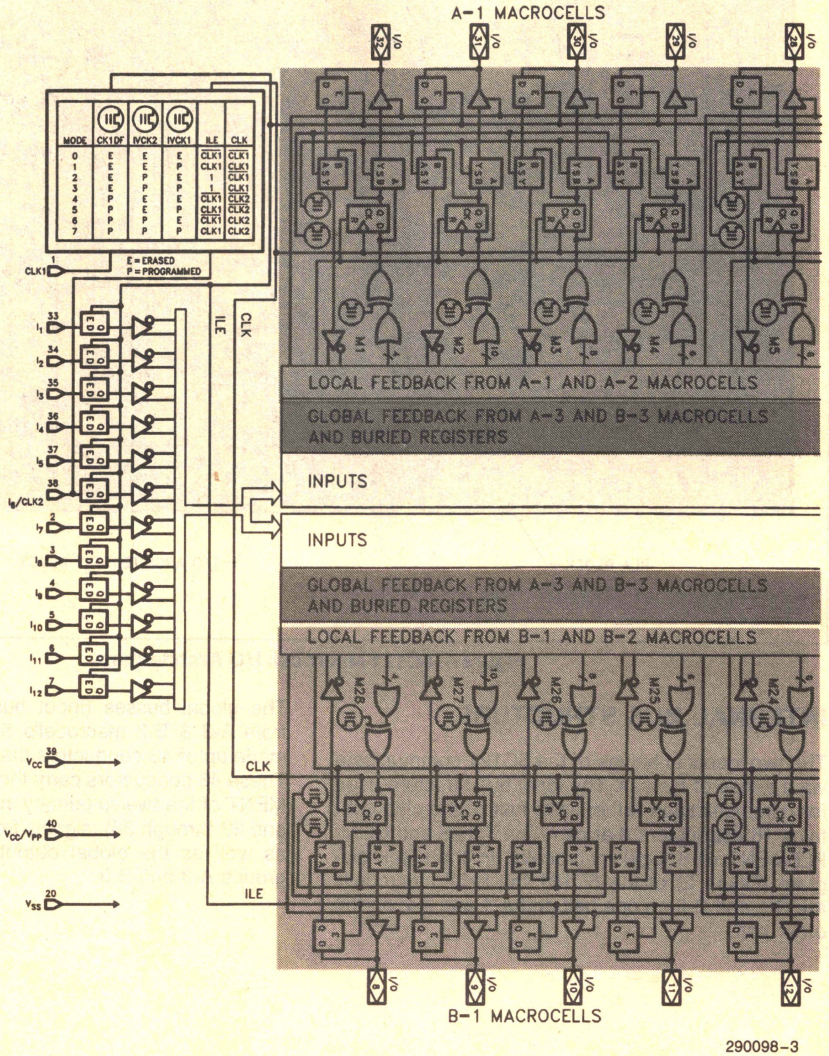


Figure 2. 5C121 Block Diagram

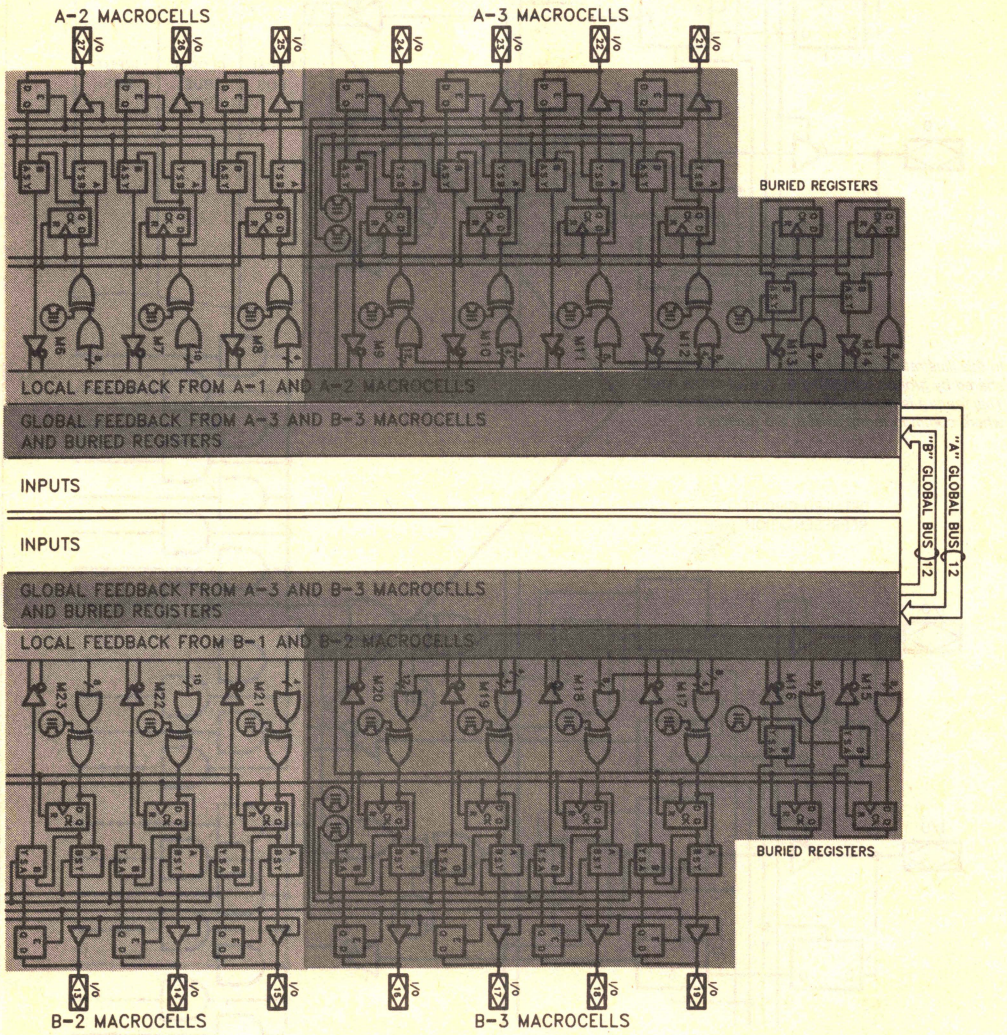


Figure 2. 5C121 Block Diagram (Continued)

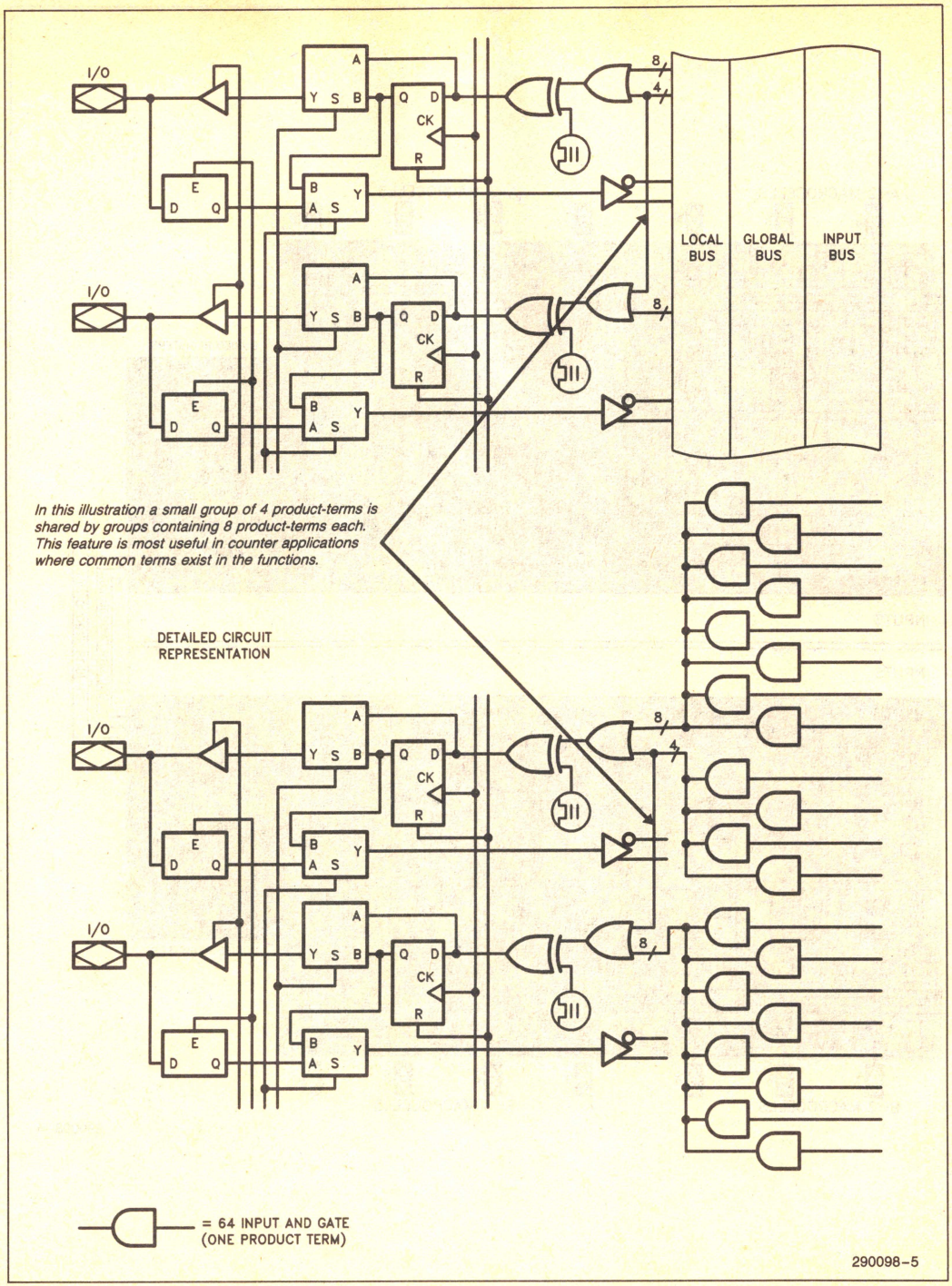


Figure 3. Shared Product-Term Circuits

SHARED PRODUCT TERMS

Macrocells 9 & 10, 11 & 12, 17 & 18 and 19 & 20 (in groups A-3 and B-3—the macrocells with global feedback) have the facility to share a total of 16 additional product terms. This sharing takes place between pairs of adjacent macrocells. This capability enables, for example, macrocells 9 and 10 to expand to 16 and 8 effective product terms respectively, and for macrocells 11 and 12 both to expand to 12 effective product terms. Figure 3 shows this sharing technique in detail. This facility is primarily of use in state machine and counter applications where common product terms are frequently required among output functions.

MACROCELL-BUS INTERFACE

As discussed earlier, the macrocells within the 5C121 are interconnected to other macrocells and inputs to the device via three internal data busses.

The product terms span the entire bus structure (local feedback, global feedback and input buses) that is adjacent to their macrocell (see Figure 4) so that they may produce a logical AND of any of the variables (or their complements) that are present on the busses.

All macrocells have the ability to return data to the local or the global bus. Feedback data may originate from the output of the macrocell or from the I/O pin. Feedback to the global bus communicates throughout the part. Macrocells that feedback to the local bus communicate only to their half of the 5C121. Connections to and from the signal busses are made with EPROM switches that provide the reprogrammable logic capability of the circuit.

Macrocells in groups A-3 and B-3 and the buried registers all have global bus connections while macrocells in groups A-1, A-2 and B-1, B-2 have only local bus connections (see Block Diagram, Figure 2). Advanced features of the Intel Programmable Logic Development System II will, if desired, automatically select an appropriate macrocell to meet both the logic requirements and the connection to an appropriate signal bus to achieve the interconnection to other macrocells.

At each intersecting point in the logic array there exists an EPROM-type programmable connection. Initially, all connections are complete. This means that both the true and complement of all inputs are connected to each product-term. Connections are opened during the programming process. Therefore any product term can be connected to the true or complement of any input. When both the true and complement connections of any input are left intact, a logical false results on the output of the AND gate. If both the true and complement connections of any input are programmed open, then a logical "don't care" results for that input. If all inputs for a product term are programmed open, then a logical true results on the output of the AND gate.

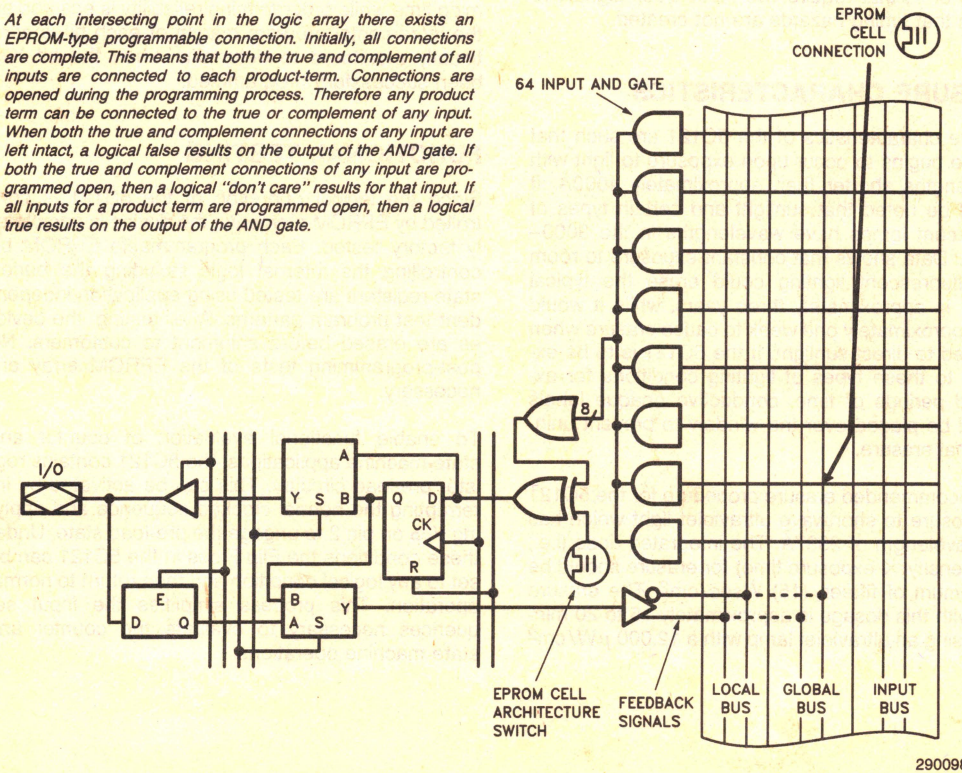


Figure 4. Macrocell-Bus Interface

CLOCK MODE CONTROL

The 5C121 contains two internal clock data paths that drive the input latches (transparent 7475 type) and the output registers. These clocks may be programmed into one of 8 operating modes (see clock mode Table 1). Figure 1 shows a typical macrocell which is driven by the master clock signal CLK and the input latch-enable signal ILE.

The master clock signal is input via pin 1. If programmed modes 4, 5, 6 & 7 are chosen, a second clock signal is required which is input via pin 38 (see Figure 5). Table 1 shows the operation of each clock programming mode.

If modes 0, 1, 4, 5, 6 or 7 are chosen (i.e. latching of the inputs is required), all inputs, both dedicated and I/O, are latched with the same ILE signal. Data applied to the inputs when CLK1 is low (high) is latched when CLK1 goes high (low) and will stay latched as long as CLK1 stays high (low). Levels shown in parenthesis are for modes 1, 5 & 7 and levels shown outside parenthesis are for modes 0, 4 & 6.

Care is required when using any of the clock modes 4, 5, 6 or 7, that require two input clock signals to ensure that timing hazards are not created.

ERASURE CHARACTERISTICS

Erasure characteristics of the 5C121 are such that erasure begins to occur upon exposure to light with wavelengths shorter than approximately 4000Å. It should be noted that sunlight and certain types of fluorescent lamps have wavelengths in the 3000–4000Å. Data shows that constant exposure to room level fluorescent lighting could erase the typical 5C121 in approximately three years, while it would take approximately one week to cause erasure when exposed to direct sunlight. If the 5C121 is to be exposed to these types of lighting conditions for extended periods of time, conductive opaque labels should be placed over the window to prevent unintentional erasure.

The recommended erasure procedure for the 5C121 is exposure to shortwave ultraviolet light which has the wavelength of 2537Å. The integrated dose (i.e., UV intensity $\times$ exposure time) for erasure should be a minimum of fifteen (15) Wsec/cm^2. The erasure time with this dosage is approximately 15 to 20 minutes using an ultraviolet lamp with a 12,000 μW/cm^2 power rating. The 5C121 should be placed within one inch of the lamp tubes during erasure. The maximum integrated dose the 5C121 can be exposed to without damage is 7258 Wsec/cm^2 (1 week @ 12,000 μW/cm^2). Exposure to high intensity UV light for longer periods may cause permanent damage.

PROGRAMMING CHARACTERISTICS

Initially, and after erasure, all the EPROM control bits of the 5C121 are connected (in the "1" state). Each of the connected control bits are selectively disconnected by programming the EPROM cell into their "0" state. Programming voltage and waveform specifications are available by request from Intel to support programming of the 5C121.

int$_e$ligent Programming™ Algorithm

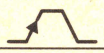
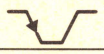
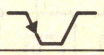
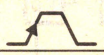
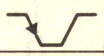
The 5C121 supports the int$_e$ligent Programming Algorithm which rapidly programs Intel H-ELPDs (and EPROMs) using an efficient and reliable method. The int$_e$ligent Programming Algorithm is particularly suited to the production programming environment. This method greatly decreases the overall programming time while programming reliability is ensured as the incremental program margin of each bit is continually monitored to determine when the bit has been successfully programmed.

FUNCTIONAL TESTING

Since the logical operation of the 5C121 is controlled by EPROM elements, the device is completely factory tested. Each programmable EPROM bit controlling the internal logic including the buried state registers are tested using application-independent test program patterns. After testing, the devices are erased before shipment to customers. No post-programming tests of the EPROM array are necessary.

To enable functional evaluation of counter and state-machine applications, the 5C121 contains register pre-load circuitry. This can be activated by interrupting the normal clocked sequence and applying V_{PP} on pin 2 to engage the pre-load state. Under these conditions the Flip Flops in the 5C121 can be set to any logical condition and then return to normal operation. This process simplifies the input sequences necessary to evaluate the counter and state machine operations.

Table 1. Clock Programming (Key: L = Latched; T = Transparent)

Programmed Mode	Input Signals Are Latched When:	Output Registers Change State When:	Clock Configuration
0	CLK1 (Pin 1)  L T	CLK1 (Pin 1)	1 Clock
1	CLK1 (Pin 1)  T L	CLK1 (Pin 1)	1 Clock
2	Inputs Not Latched	CLK1 (Pin 1)	1 Clock
3	Inputs Not Latched	CLK1 (Pin 1)	1 Clock
4	CLK1 (Pin 1) L T	CLK2 (Pin 38)	2 Clocks
5	CLK1 (Pin 1)  T L	CLK2 (Pin 38)	2 Clock
6	CLK1 (Pin 1)  L T	CLK2 (Pin 38)	2 Clocks
7	CLK1 (Pin 1)  T L	CLK2 (Pin 38)	2 Clocks

DESIGN RECOMMENDATIONS

For proper operation it is recommended that input and output pins be constrained to the range GND < (V_{IN} or V_{OUT}) < V_{CC}. Unused inputs should be tied to an appropriate logic level (e.g. either V_{CC} or GND) to minimize device power consumption.

When utilizing a macrocell with an I/O pin connection as a buried macrocell (i.e. just using the macrocell for feedback purposes to other macrocells), its I/O pin is a 'reserved pin'. (The Intel Programmable Logic Development System II will label the pin 'RESERVED' in the utilization report that it generates.) Such an I/O pin will actually be an output pin and should not be grounded. It should be left unconnected such that it can go high or low depending on the state of the macrocell's output.

In normal operation V_{CC}/V_{PP} (pin 40) should be connected directly to V_{CC} (pin 39).

DESIGN SECURITY

A single EPROM bit provides a programmable design secruity feature that controls the access to the data programmed into the device. If this bit is set, a proprietary design within the device cannot be copied. This EPROM security bit enables a higher degree of design security than fused-based devices since programmed data within EPROM cells is invisible even to microscopic evaluation. The EPROM security bit, along with all the other EPROM control bits, will be reset by erasing the device.

LATCH-UP IMMUNITY

All of the input, I/O, and clock pins of the 5C121 have been designed to resist latch-up which is inherent in inferior CMOS structures. The 5C121 is designed with Intel's proprietary CHMOS II-E EPROM process. Thus, each of the 5C121 pins will not experience latch-up with currents up to 100 mA and voltages ranging from $-1V$ to $V_{CC} + 1V$. Furthermore, the programming pin is designed to resist latch-up to the 13.5V maximum device limit.

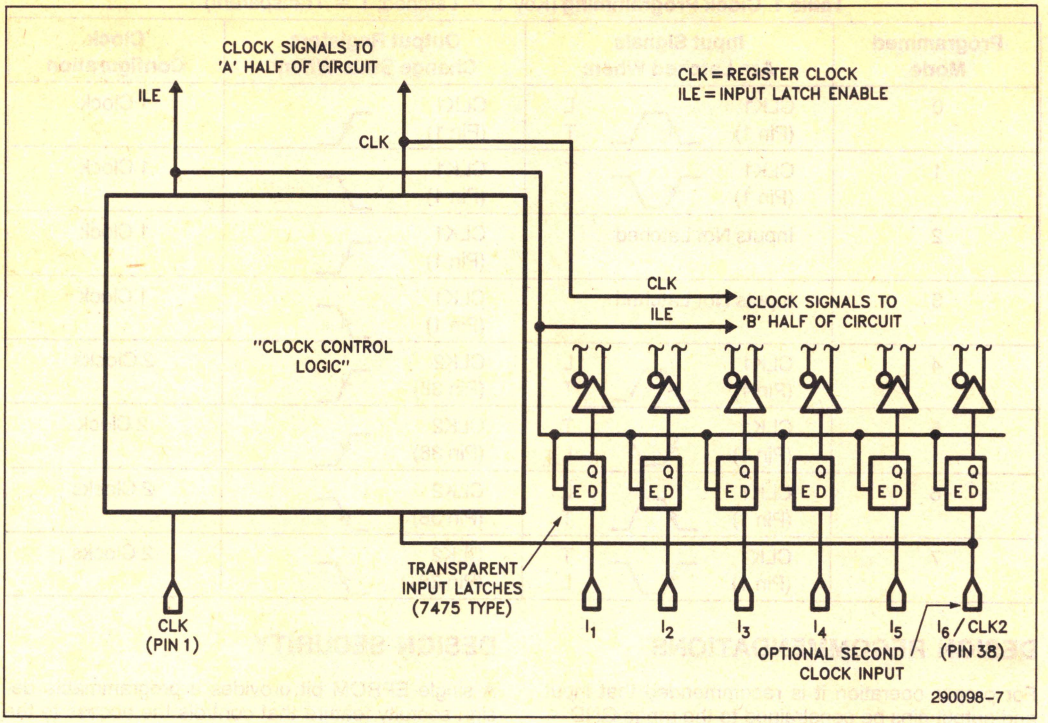


Figure 5. Programmable Clock Control System

ABSOLUTE MAXIMUM RATINGS*

Symbol	Parameter	Min	Max	Unit
V_{CC}	Supply Voltage(1)	−2.0	7.0	V
V_{PP}	Programming Supply Voltage(1)	−2.0	13.5	V
V_I	DC Input Voltage(1)(2)	−0.5	$V_{CC}+0.5$	V
I_{CC}	DC V_{CC} Current(4)		100	mA
T_{stg}	Storage Temperature	−65	+150	°C
T_{amb}	Ambient Temperature(3)	−10	+85	°C

Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

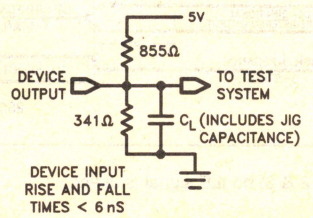
NOTES:
1. Voltages with respect to ground.
2. Minimum DC input is −0.5V. During transitions, the inputs may undershoot to −2.0V for periods less than 20 ns under no load conditions.
3. Under bias.
4. With outputs tristated.

D.C. CHARACTERISTICS T_A = 0° to 70°C, V_{CC} = 5.0V ±5%

Symbol	Parameter	Conditions		Min	Typ	Max	Unit
V_{IH}	HIGH Level Input Voltage			2.0		$V_{CC}+0.3$	V
V_{IL}	LOW Level Input Voltage			−0.3		0.8	V
V_{OH}	HIGH Level Output Voltage	I_O = −4.0 mA DC		2.4			V
V_{OL}	LOW Level Output Voltage	I_O = 4.0 mA DC				0.45	V
I_I	Input Leakage Current	$V_I = V_{CC}$ or GND				±10.0	µA
I_{OZ}	3-State Output Off-State Current	$V_O = V_{CC}$ or GND				±10.0	µA
I_{SB}	V_{CC} Supply Current (Standby) (Note 6)	$V_I = V_{CC}$ or GND, I_O = 0	CMOS Inputs			3	mA
			TTL Inputs			30	
I_{CC}	V_{CC} Supply Current (Active)	No Load, f = 10 MHz	CMOS Inputs			50	mA
			TTL Inputs			100	
I_{OS}	Output Short Circuit Current	(Note 5)				130	mA

NOTES:
5. Output shorted for no more than 1 sec. and no more than one output shorted at a time. I_{OS} is sampled but not 100% tested.
6. Chip automatically goes into standby mode if logic transitions do not occur. (Approximately 100 ns after last transition.)

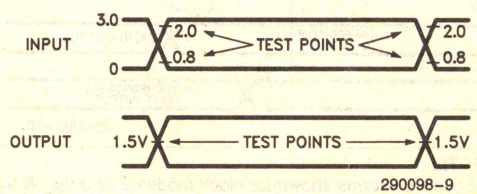
A.C. TESTING LOAD CIRCUIT



290098-8

C_L = 50 pF

A.C. TESTING INPUT, OUTPUT WAVEFORM



290098-9

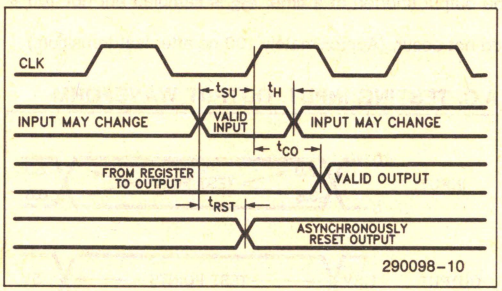
A.C. Testing: Inputs are Driven at 3.0V for a Logic "1" and 0V for a Logic "0". Timing Measurements are made at 2.0V for a Logic "1" and 0.8V for a Logic "0" on inputs. Outputs are measured at a 1.5V point.

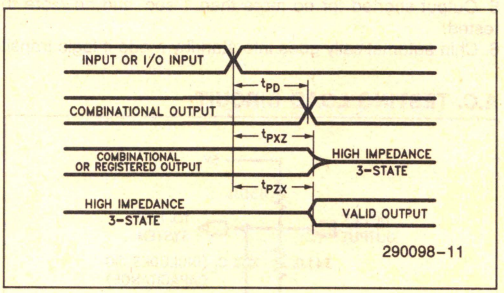
A.C. CHARACTERISTICS T_A = 0° to 70°C, V_{CC} = 5.0V ±5%

Symbol	Parameter	Device Conditions	5C121-50		5C121-65		5C121-90		Unit
			Min	Max	Min	Max	Min	Max	
t_{PD}	Non-Registered Input or I/O Input to Non-Registered Output			50		65		90	ns
t_{PZX}(7)	Non-Registered Input or I/O Input to Output Enable	C_L = 30 pF		50		65		90	ns
t_{PXZ}(7)	Non-Registered Input or I/O Input to Output Disable			50		65		90	ns
t_{SU}	Non-Registered Input or I/O Input to Output Register Setup		37		47		62		ns
t_H	Non-Registered Input or I/O Input to Output Register Hold		0		0		0		ns
t_{CH}	Clock High Time		20		25		30		ns
t_{CL}	Clock Low Time	C_L = 30 pF	20		25		30		ns
t_{CO}	Clock to Output Delay			28		33		38	ns
t_{CNT}	Minimum Clock Period (Register Output Feedback to Register Input—Internal Path)			50		55		75	ns
f_{CNT}	Maximum Frequency ($1/t_{CNT}$)		20.0		18.0		13.0		MHz
f_{MAX}	Maximum Frequency ($1/t_{SU}$)		25.0		20		16.0		MHz
t_{RST}	Asynchronous Reset Time			50		65		90	ns
t_{ILS}	Set Up Time for Latching Inputs		0		0		0		ns
t_{ILH}	Hold Time for Latching Inputs		15		20		25		ns
t_{C1C2}	Minimum Clock 1 to Clock 2 Delay		40		50		65		ns
t_{ILDFS}	Input Latch to D-FF Setup Time	Mode 0, 1	40		50		65		ns
t_{DFILS}	D-FF to Input Latch Setup Time		25		30		35		ns
t_{P3}	Minimum Period for a 2-Clock System (T_{C1C2} + t_{CO1})			65		85		100	ns
f_3	Maximum Frequency ($1/t_{P3}$)		15.0		12.0		10.0		MHz

NOTE:
7. t_{PZX} and t_{PXZ} are measured at ±0.5V from steady state voltage as driven by spec. output load. t_{PXZ} is measured with C_L = 5 pF.

SWITCHING WAVEFORMS

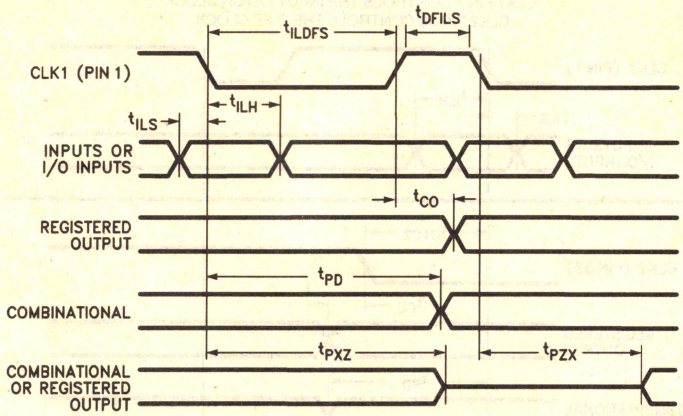




NOTE:
Above waveforms shown for clock modes 2 or 3 (t_{SU} & t_H are as in modes 2 & 3; no ILE signal is used).

CLOCK MODES SWITCHING WAVEFORMS

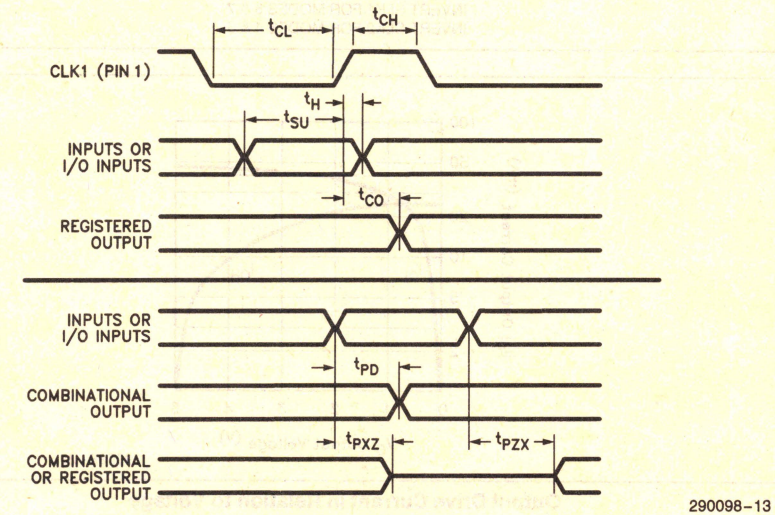
1-CLOCK SYSTEM: MODES 0 AND 1



290098-12

INVERT CLK1 FOR MODE 0

1-CLOCK SYSTEM: MODES 2 AND 3

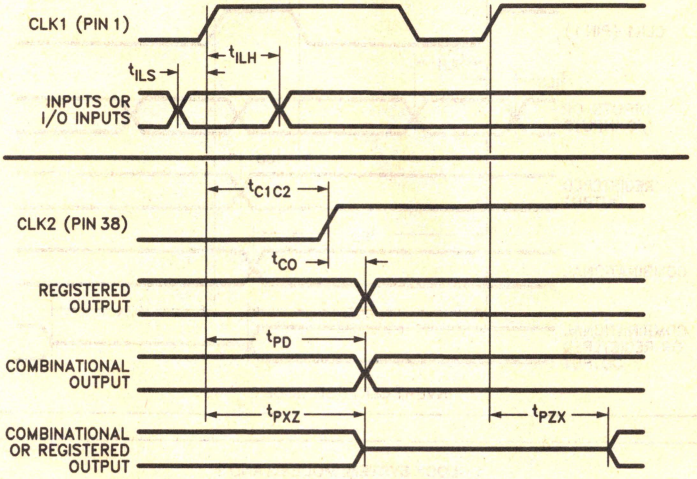


290098-13

INVERT CLK1 FOR MODE 2

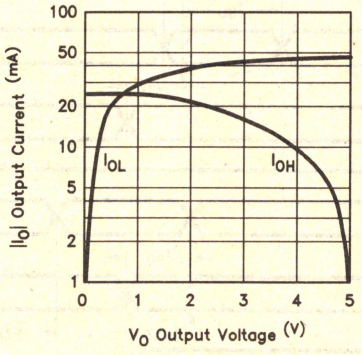
CLOCK MODES
SWITCHING WAVEFORMS (Continued)

2-CLOCK SYSTEMS: MODES 4 THROUGH 7
CLK1 PIN 1 CONTROLS THE INPUT LATCH CLOCK
CLK2 PIN 38 CONTROLS THE D-FF CLOCK.



290098-14

INVERT CLK1 FOR MODES 5 & 7
INVERT CLK2 FOR MODES 4 & 5



290098-20

Output Drive Current in Relation to Voltage

Intel Programmable Logic Development System II (iPLDS II)

The iPLDS II provides all the tools needed to design with Intel H-Series EPLDs or compatible devices (see Figure 6). It contains comprehensive third generation software that supports four different design entry methods, minimizes logic, does automatic pin assignments and produces the best design fit for the selected EPLD. It is user friendly with guided menus, on-line Help messages and soft key inputs.

In addition, the iPLDS II contains programmer hardware in the form of an expansion card for the PC with programming software to enable the user to program EPLDs, read and verify programmed devices and also to graphically edit programming files. The software generates industry standard JEDEC object code output files which can be downloaded to other programmers as well.

The iPLDS II has interfaces to popular schematic capture packages (Dash series from Futurenet* and PC CAPS** from PCAD) to enable designs to be entered using schematics. A more integrated schematic entry method is provided by SCHEMA II-PLD, a low-cost schematic capture package that supports EPLD primitives and user-defined macro symbols. SCHEMA II-PLD contains the EPLD Design Manager, which provides a single user interface to both SCHEMA II-PLD and iPLS II software. The other design entry formats supported are Boolean equation entry and State Machine design entry.

The iPLDS II runs on the IBM† PC, PC/XT or PC/AT and other compatible machines with the following configuration:

(1) At least one floppy disk drive and hard disk drive

(2) MS-DOS†† Operating System Version 2.0 or later release

(3) 640K Memory

(4) Intel iUP-PC Universal Programmer-Personal Computer and GUPI Adaptor (supplied with iPLDS II).

Detailed information on the Intel Programmable Logic Development System II is contained in a separate Intel data sheet (Order Number: 280168).

*FutureNet is a registered trademark of FutureNet Corporation. DASH is a trademark of FutureNet Corporation.

**PC-CAPS is a trademark of P-CAD Corporation.

†IBM Personal Computer is a registered trademark of International Business Machine Corporation.

††MS-DOS is a registered trademark of Microsoft Corporation.

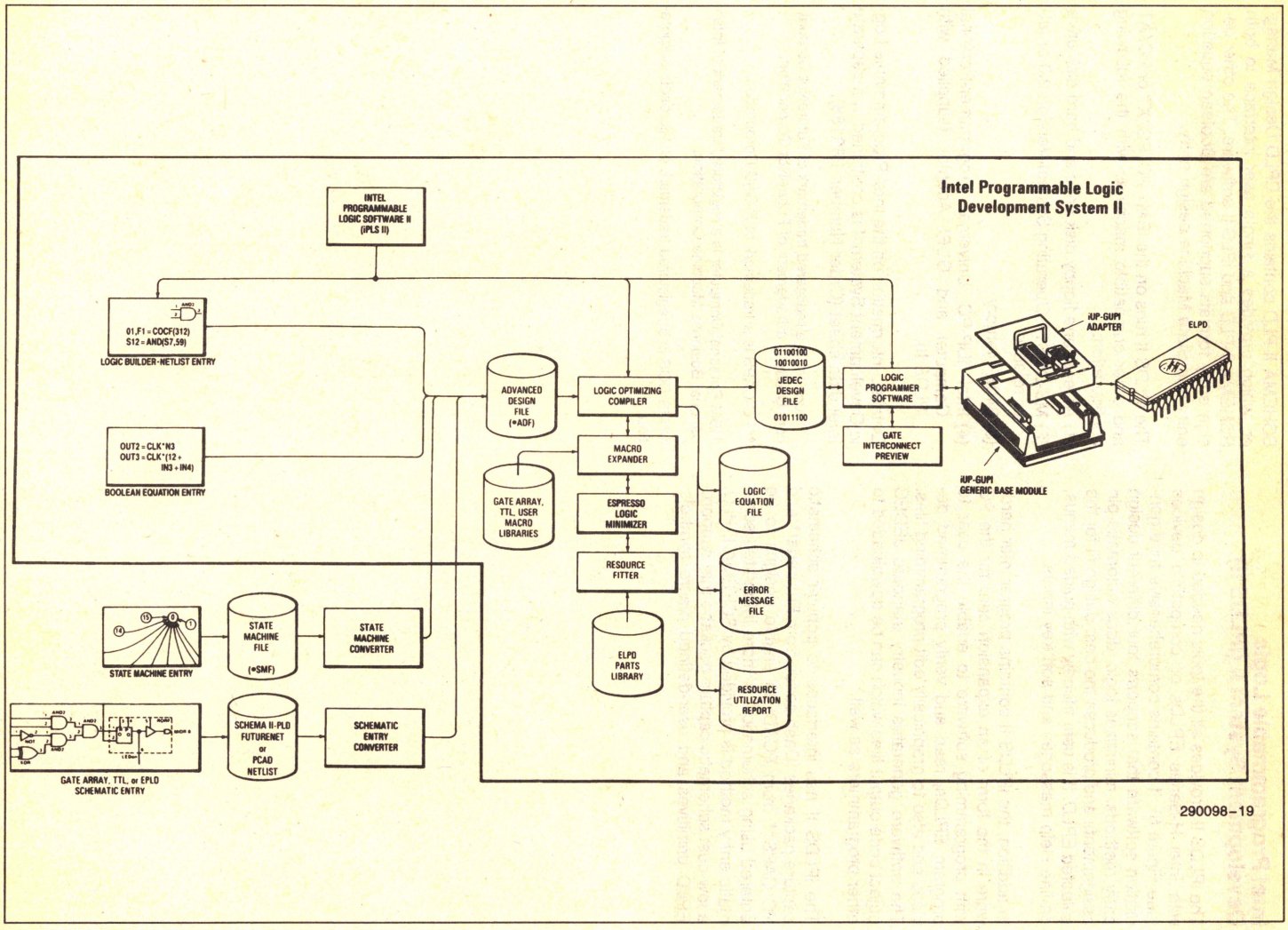


Figure 6. Intel Programmable Logic Development System II

PRELIMINARY

5C180
1800-GATE CHMOS ERASABLE PROGRAMMABLE LOGIC DEVICE

- **High Performance LSI Semicustom Logic Replacement for TTL and 74HC SSI and MSI Logic**
- **CHMOS EPROM Technology-Based UV Erasable**
- **48 Macrocells with Programmable I/O Architecture; up to 64 Inputs (16 Dedicated, 48 I/O) or 48 Outputs**
- **High Speed t_{PD} (max) 75 ns Operating Frequency (Max) 12 MHz**
- **Low Power; 100 μW Typical Standby Dissipation**
- **Programmable "Security Bit" Allows Total Protection of Proprietary Designs**
- **Dual Feedback Signals Allowing I/O Pins to Be Used for Buried Logic and Dedicated Input**
- **Programmable Clock System with Four Synchronous Clocks as well as Asynchronous Clocking Option on All Registers**
- **Programmable Registers. Can Be Configured as D, T, SR or JK Types with Individual Reset Controls**
- **Register Pre-Load and Erasable Array for 100% Generic Testability**
- **68-Pin J-Lead Chip Carrier and Pin Grid Array Packages**

(See packaging spec., Order #231369)

The Intel 5C180 EPLD (Erasable Programmable Logic Device) is a CHMOS LSI Logic Device capable of integrating 1800 to over 2000 equivalent gates of SSI/MSI logic. This user customizable Logic Device is available in a 68-pin J-Leaded chip carrier or Pin Grid Array package and has the benefits of low power and increased flexibility.

The 5C180 EPLD uses CHMOS EPROM (floating gate) cells as logic control elements instead of fuses. Use of Intel's advanced CHMOS II-E EPROM process technology enables greater logic densities to be achieved with superior speed and power performance. The EPROM technology also enables these devices to be 100% factory tested by the programming and the erasure of all the EPROM logic control elements in the device.

The architecture of the 5C180 is based on the "Sum of Products" PLA (Programmable Logic Array) structure with a programmable AND array feeding into a fixed OR array. The 48 macrocells of the 5C180 can be partitioned into 4 identical quandrants each containing 12 macrocells. This device makes use of a segmented PLA structure with local and global bus structures to provide for increased performance and greater device utilization. The 5C180 has unique architectural features that allow programming of all 48 registers to D, T, SR or JK configurations without sacrificing product terms. These registers can be either clocked asynchronously or in banks with four synchronous clocks. In addition, the 16 global macrocells have two independent feedback paths to the array that allow for buried logic implementation together with use of the I/O pin for input functions.

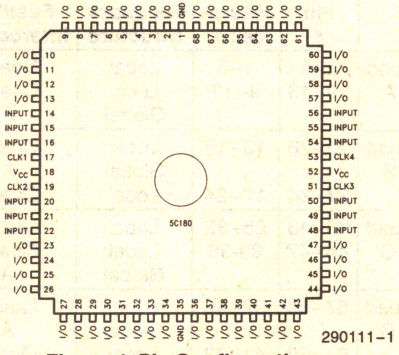


Figure 1. Pin Configuration

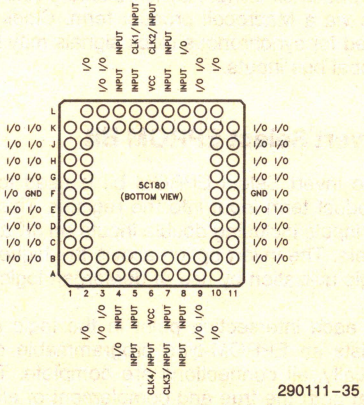


Figure 2. PGA Pin Configuration

November 1987
Order Number: 290111-004

ARCHITECTURE DESCRIPTION

Externally, the 5C180 provides 12 dedicated data inputs, 4 synchronous clock inputs, and 48 I/O pins which may be individually programmed for input, output, or bi-directional operation.

The Block Diagram is shown in Figure 2. The internal architecture is organized in familiar sum-of-products (AND-OR) structure. The 5C180 houses a total of 480 product terms distributed among 48 Macrocells. The basic Macrocell structure is shown in Figure 3. Input and feedback signals are selectively connected to product terms via EPROM cells. The output of the AND array feeds a fixed OR gate to produce sum-of-products logic. The final output may be combinatorial or registered, programmed active high or low. Combinatorial, registered, or pin feedback is also user-defined.

The 5C180 is portioned into 4 identical quadrants. Each quadrant contains 12 Macrocells. Input signals to the Macrocells come from the 5C180 Local and Global bus structures. These two buses comprise an 88-input AND array for each quadrant. The output of each Macrocell feeds an I/O Architecture Control Block which contains output and feedback selection.

Four dedicated clock inputs provide synchronous clock signals to the 5C180 internal registers. There is one synchronous clock per quadrant. Therefore each clock signal controls a bank of 12 registers. CLK1 may be connected to registers in Macrocells 1–12, CLK2 with Macrocells 13–24, CLK3 with Macrocells 25–36, and CLK4 with Macrocells 37–48. With synchronous clocks, the flip-flops are positive edge triggered. Both true and complement signals for each dedicated clock input may also be used within the AND array. All 48 internal registers may be individually programmed for synchronous or asynchronous clocking. Asynchronous clocking is possible via a Macrocell product term. Clock inputs not used for synchronous clock signals may be used as global bus inputs.

Invert Select EPROM Bit

The Invert Select EPROM bit is used to invert the product term input into the register. This applies to all inputs including double inputs on JK and SR registers. The invert option allows the highest possible logic utilization by use of deMorgan logic inversion.

At each intersecting point in the logic array there exists an EPROM-type programmable connection. Initially, all connections are complete. This means that both the true and complement of all inputs are connected to each product-term. Connections are opened during the programming process. Therefore any product term can be connected to the true or complement of any input. When both the true and complement connections of any input are left intact, a logical false results on the output of the AND gate. If both the true and complement connections of any input are programmed open, then a logical "don't care" results for that input. If all inputs for a product term are programmed open, then a logical true results on the output of the AND gate.

BUS STRUCTURE

Input and feedback signals are connected to each 5C180 Macrocell via a Local and Global Bus. Figure 4 shows the Macrocell-Bus interface for Quadrant D. The Global Bus contains 64 input signals while the Local Bus has 24.

Within the 5C180 Macrocell, the product-terms share the entire bus structure. Therefore, a logical AND of any of the variables (or their complements) that is present on the buses may be produced by each product term.

All quadrants share the same Global Bus. Inputs to the bus come from the true and complement signals of the 12 dedicated data inputs, 4 clock inputs, and the 16 Global Macrocell pin feedback signals.

Each quadrant has its own Local Bus. Inputs to this bus come from the 12 quadrant Macrocells. For the eight Local Macrocells, the signals can be either from the Macrocell internal logic or from the pin. For the four Global Macrocells, the signals come from the Macrocell internal logic only.

Table 1 summarizes the Macrocell interconnect.

Table 1. Macrocell Interconnect

	Pin #	Macrocell #	Feedback Structure	Feedback Interconnect
Quad A	2–9	1–8	Local	Quad A
	10–13	9–12	Local	Quad A
			Global	All
Quad B	23–26	13–16	Local	Quad B
			Global	All
	27–34	17–24	Local	Quad B
Quad C	36–43	25–32	Local	Quad C
	44–47	33-36	Local	Quad C
			Global	All
Quad D	57–60	37–40	Local	Quad D
			Global	All
	61–68	41–48	Local	Quad D

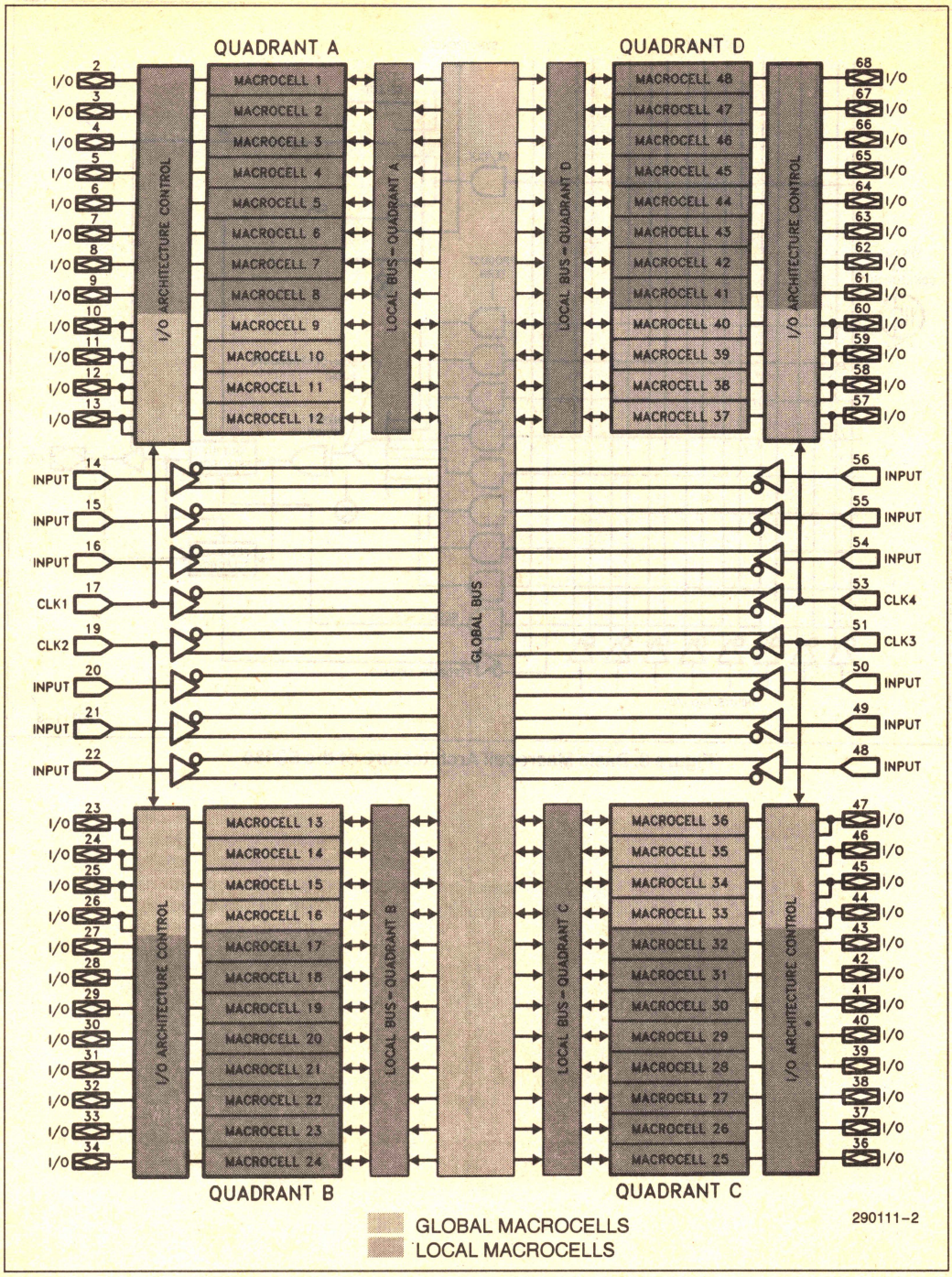


Figure 2. 5C180 Block Diagram

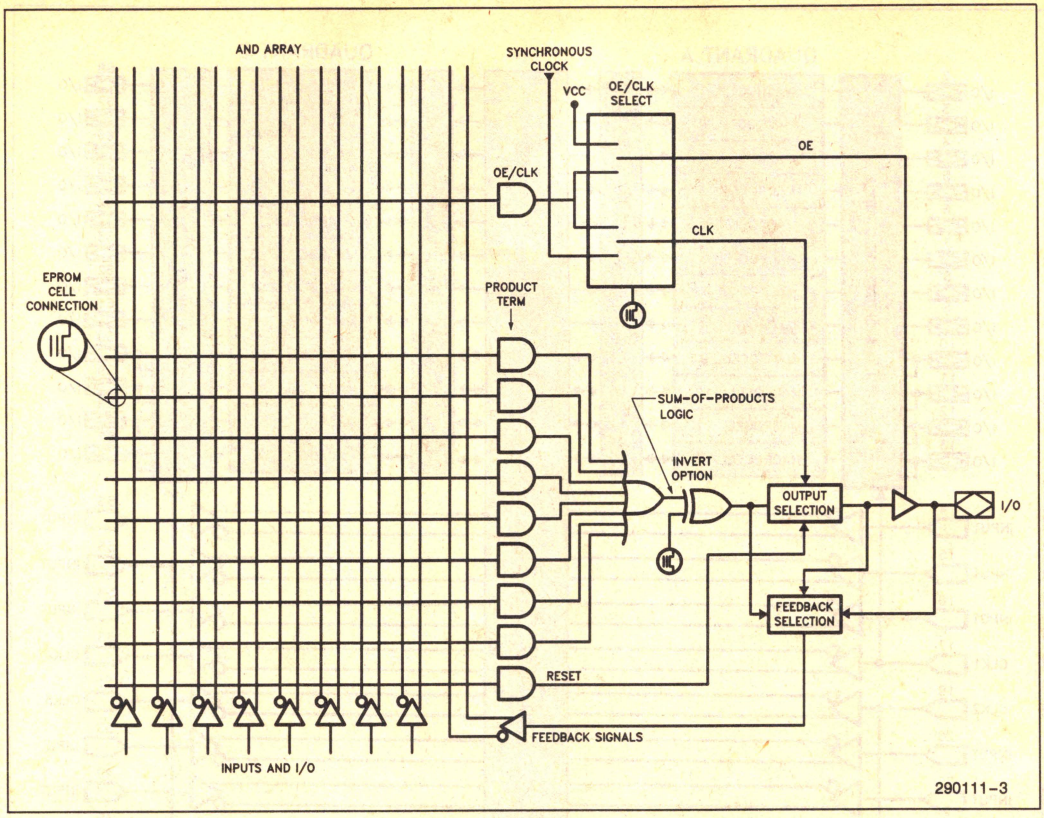


Figure 3. Basic Macrocell Architecture of the 5C180

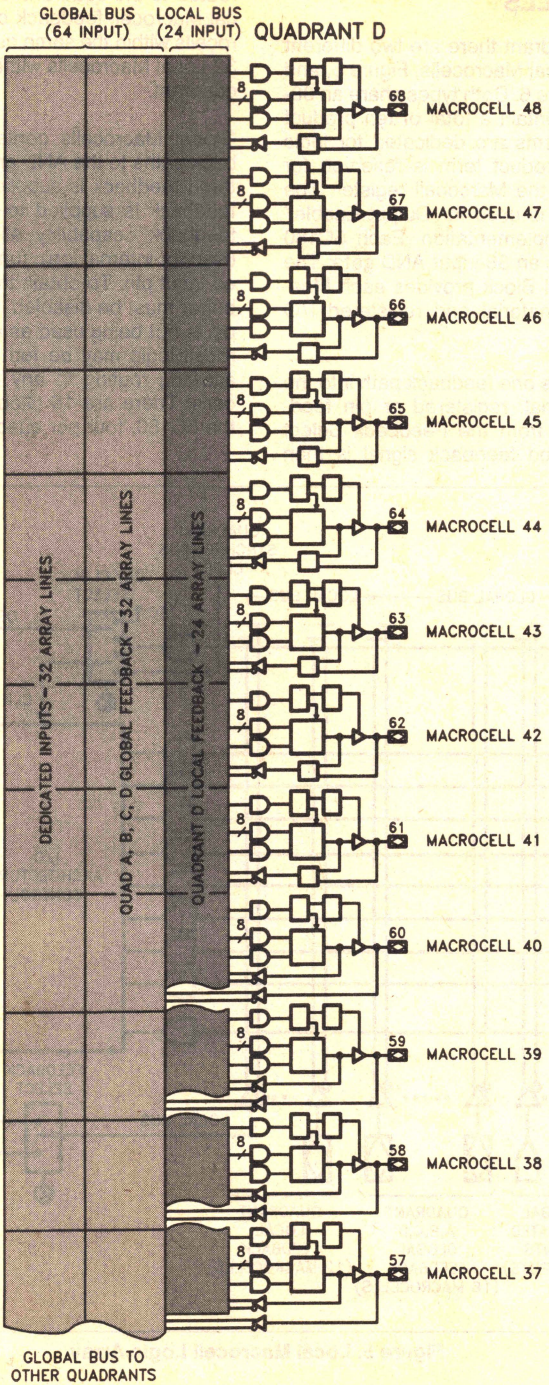


Figure 4. Quadrant "D" Bus Interface

5C180 MACROCELLS

Within each 5C180 quadrant there are two different types of Macrocells; Local Macrocells, Figure 5, and Global Macrocells, Figure 6. Both types share an 88-input AND array and contain a total of ten product terms. Eight product terms are dedicated for logic implementation. One product term is reserved for Asynchronous Clear to the Marcocell register. The remaining product term is used for Output Enable/ Asynchronous Clock implementation. Each 5C180 product term represents an 88-input AND gate. The I/O Architecture Control Block provides each Macrocell with both combinatorial and registered I/O configurations.

Local Macrocells provide one feedback path into the AND array. Combinatorial, registered or pin feedback may be selected from the Feedback Select Multiplexer. The selected feedback signal is then routed to the quadrant local bus. Therefore, the Local Macrocell feedback communicates only to Macrocells within the same quadrant. There are a total of 32 Local Macrocells within the 5C180, with eight per quadrant.

Global Macrocells contain two independent feedback paths to the AND array. Combinatorial or registered feedback is supplied to the local bus and pin feedback is supplied to the global bus. The "dual feedback" capability allows the Macrocell to be used for internal logic functions as well as a dedicated input pin. To obtain this configuration, the output buffer must be disabled. If the Global Macrocell I/O pin is not being used as a dedicated input, the Macrocell logic may be fed back along the global bus allowing routing to any of the 5C180's 48 Macrocells. There are 16 Global Macrocells contained in the 5C180, four per quadrant.

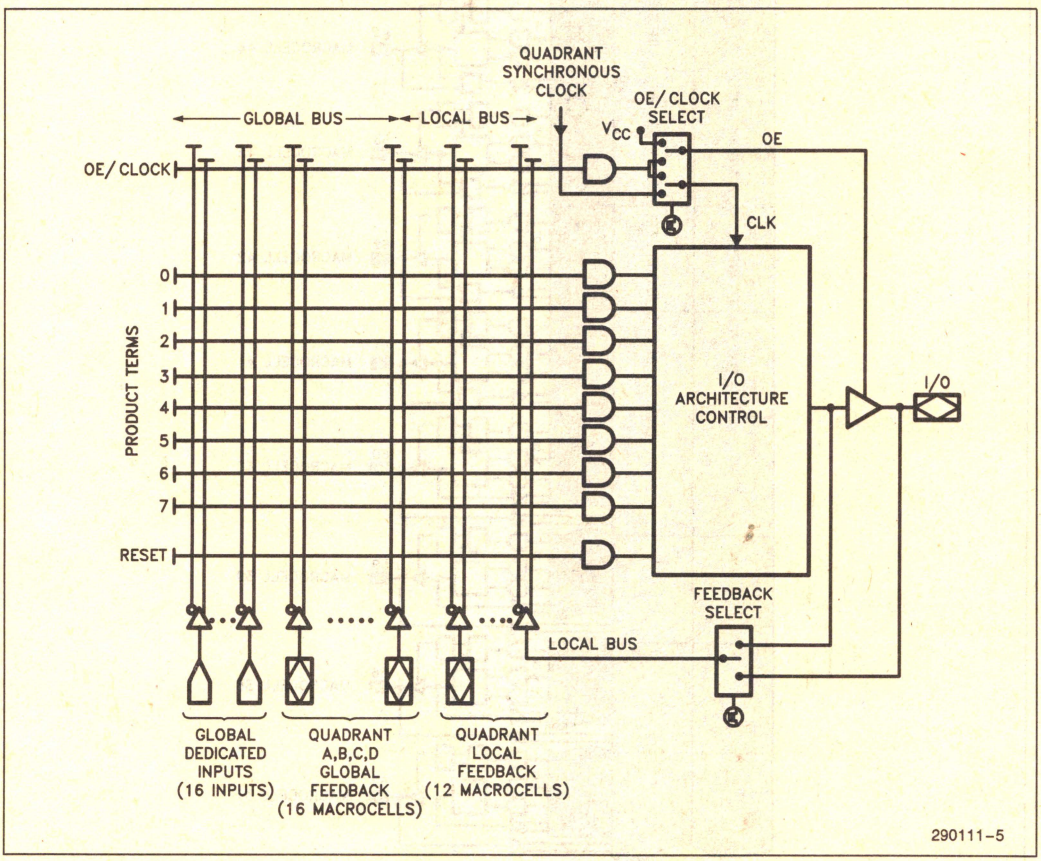


Figure 5. Local Macrocell Logic Array

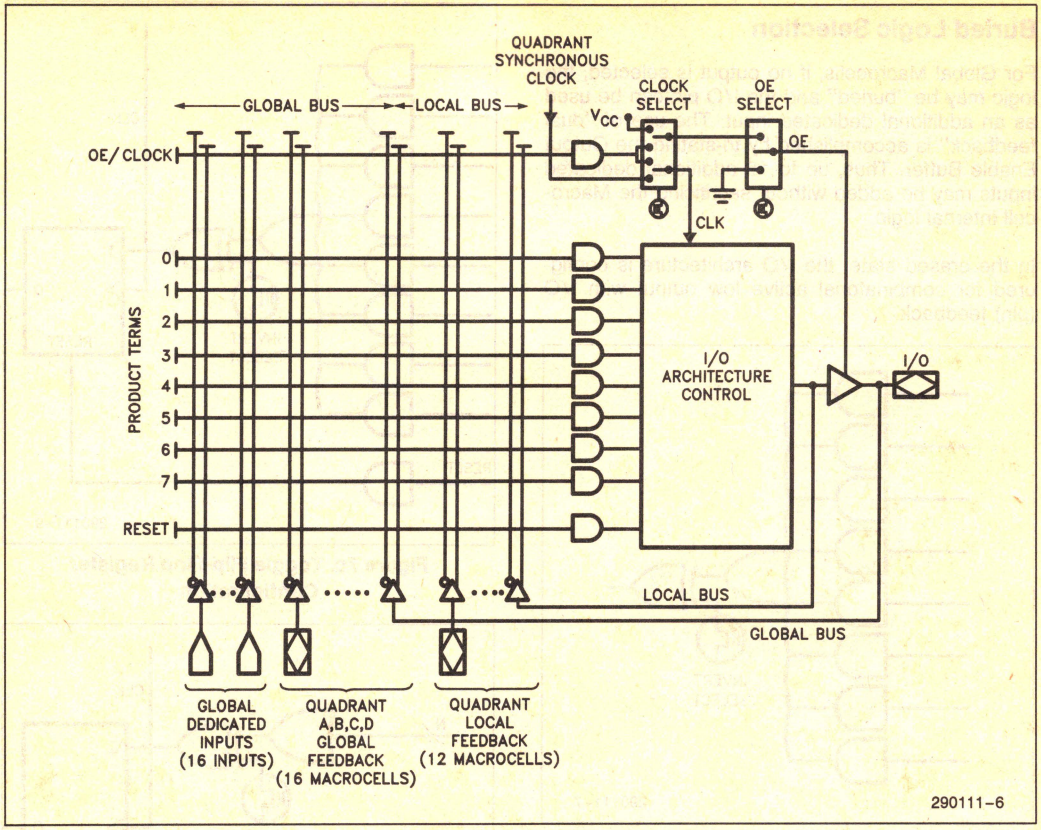


Figure 6. Global Macrocell Logic Array

MACROCELL LOGIC CONFIGURATIONS

Combinatorial Selection

In the Combinatorial configuration, eight product terms are ORed together to generate the output signal. The Invert Select EPROM bit controls output polarity and the Output Enable buffer is product-term controlled. The Feedback Select allows the user to choose combinatorial, I/O (pin) or no feedback to the respective local and global buses.

REGISTER SELECTION

The advanced I/O architecture of the 5C180 allows four different register types along with combinatorial output as illustrated in Figures 7a–7e. The register types include a T, D, JK, or SR Flip-Flop and each Macrocell I/O structure may be independently configured. In addition, all registers have an individual asynchronous RESET control from a dedicated product term derived in the AND array. When this dedicated product term is a logical one, the Macrocell register is immediately cleared to a logical zero independent of the register clock. The RESET function occurs automatically on power-up.

The four different register types shown in Figures 7b–7e are described below:

D- or T-type Flip-Flops

When either a D- or T-type Flip-Flop is configured as part of the I/O structure, all eight of the product terms into the Macrocell are ORed together and fed into the register input.

JK or SR Registers

When either a JK or SR register is configured, the eight product terms are shared among two OR gates (one for the J or S input and the other for the K or R input). The allocation for these product terms for each of the register inputs is optimized by the iPLDS II development software.

Buried Logic Selection

For Global Macrocells, if no output is selected, the logic may be "buried" and the I/O pin can be used as an additional dedicated input. The use of "dual feedback" is accomplished by tri-stating the Output Enable Buffer. Thus, up to 16 additional dedicated inputs may be added without sacrificing the Macrocell internal logic.

In the erased state, the I/O architecture is configured for combinatorial active low output with I/O (pin) feedback.

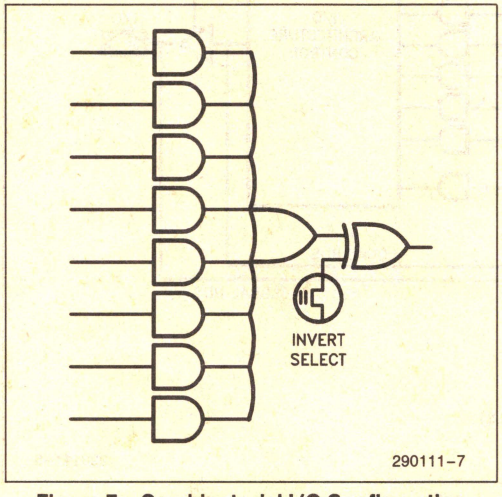


Figure 7a. Combinatorial I/O Configuration

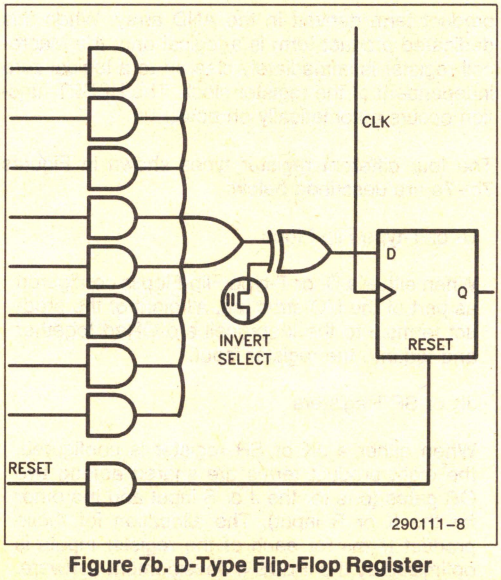


Figure 7b. D-Type Flip-Flop Register Configuration

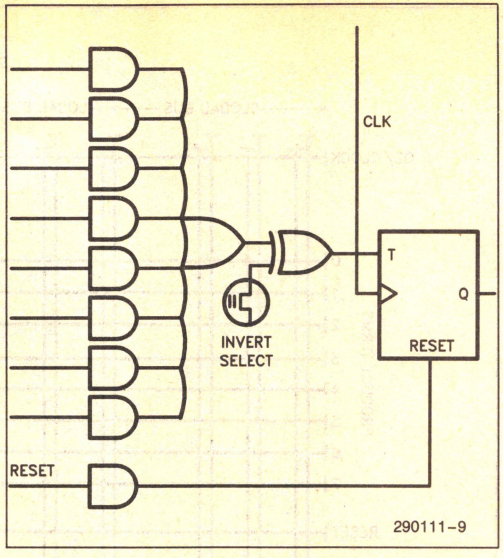


Figure 7c. Toggle Flip-Flop Register Configuration

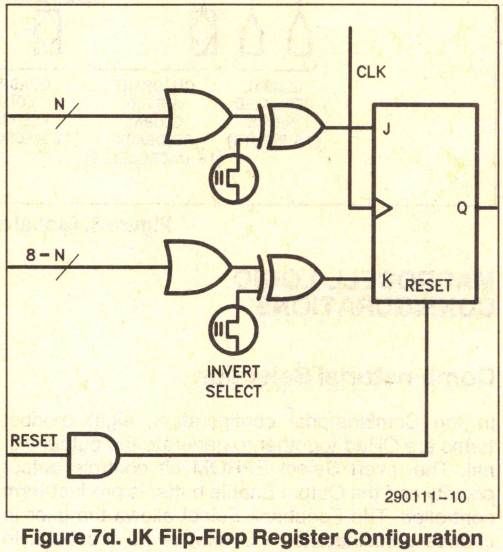


Figure 7d. JK Flip-Flop Register Configuration

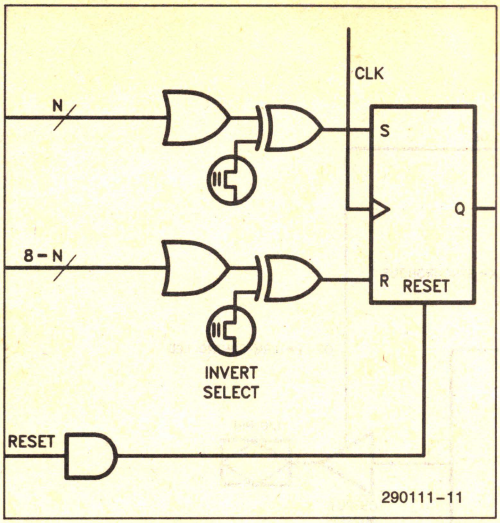


Figure 7e. SR Flip-Flop Register Configuration

MACROCELL OE/CLK SELECT

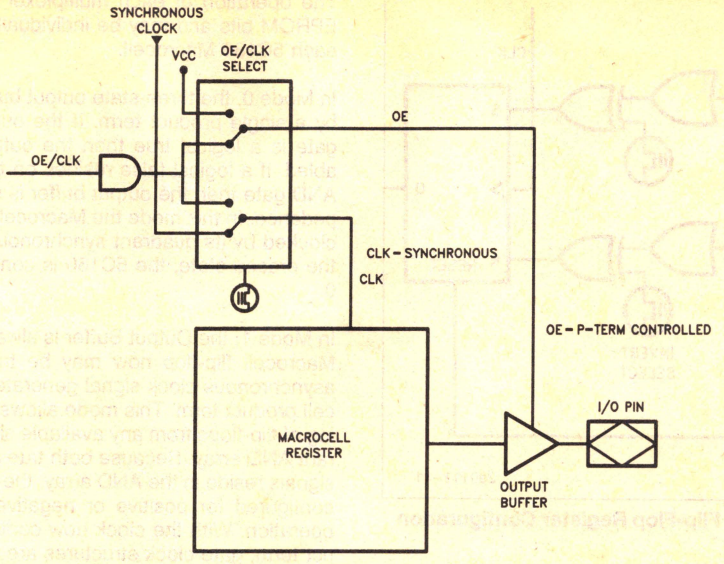
Each 5C180 register may be clocked synchronously or asynchronously. Figure 8a and 8b shows the modes of operation provided by the OE/CLK Select Multiplexers for both Local and Global Macrocells.

The operation of each multiplexer is controlled by EPROM bits and may be individually configured for each 5C180 Macrocell.

In Mode 0, the three-state output buffer is controlled by a single product term. If the output of the AND gate is a logical true then the output buffer is enabled. If a logical false resides on the output of the AND gate then the output buffer is seen as high impedance. In this mode the Macrocell flip-flop may be clocked by its quadrant synchronous clock input. In the erased state, the 5C180 is configured as Mode 0.

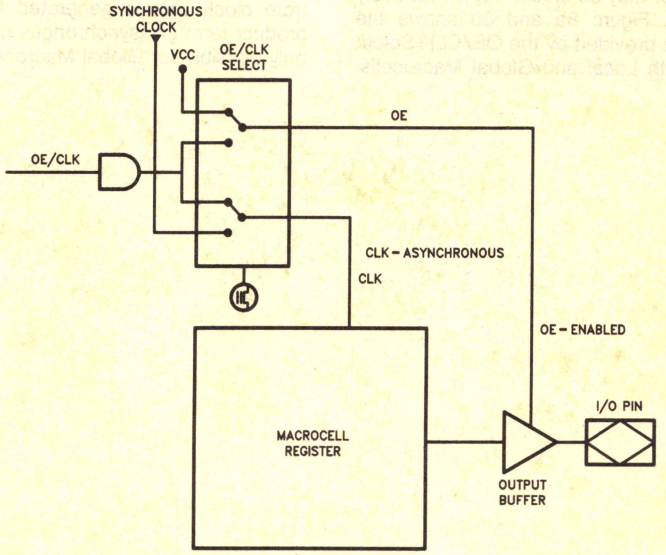
In Mode 1, the Output Buffer is always enabled. The Macrocell flip-flop now may be triggered from an asynchronous clock signal generated by the Macrocell product term. This mode allows individual clocking of flip-flops from any available signal in the quadrant AND array. Because both true and complement signals reside in the AND array, the flip-flops may be configured for positive or negative edge triggered operation. With the clock now controlled by a product term, gate clock structures are also possible.

In Modes 2 and 3, the Output Buffer is always disabled. The Macrocell flip-flop may still be triggered from clock signals generated from the Macrocell product term or asynchronous clocks. This mode is only possible for Global Macrocells.



290111-12

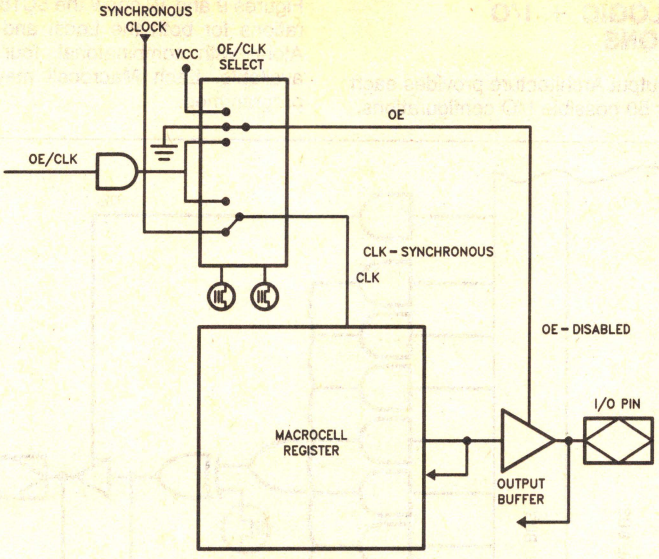
The register is clocked by the quadrant synchronous clock signal which is common to 11 other Macrocells. The output is enabled by the logic from the product term.



290111-13

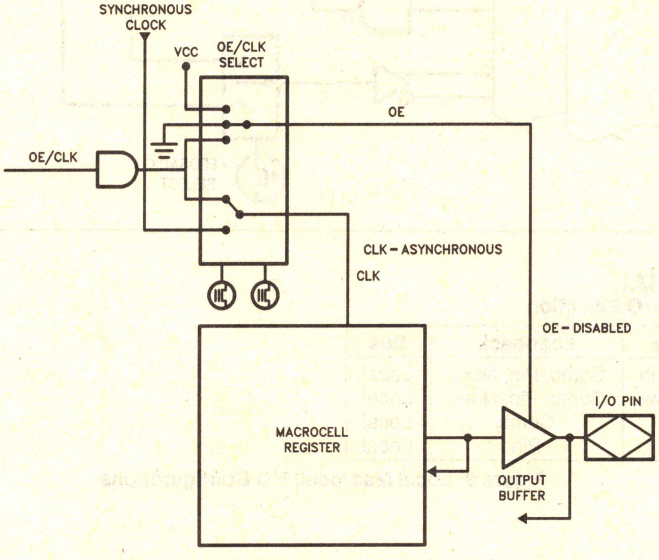
The output is permanently enabled and the register is clocked via the product term. This allows for gated clocks that may be generated from elsewhere in the 5C180.

Figure 8a. Local Macrocell OE/CLK Selection



290111-14

The output is permanently disabled and the register clocked by the quadrant synchronous clock signal. The pin can be used as an input while the register or combinational output can be fed back.



290111-15

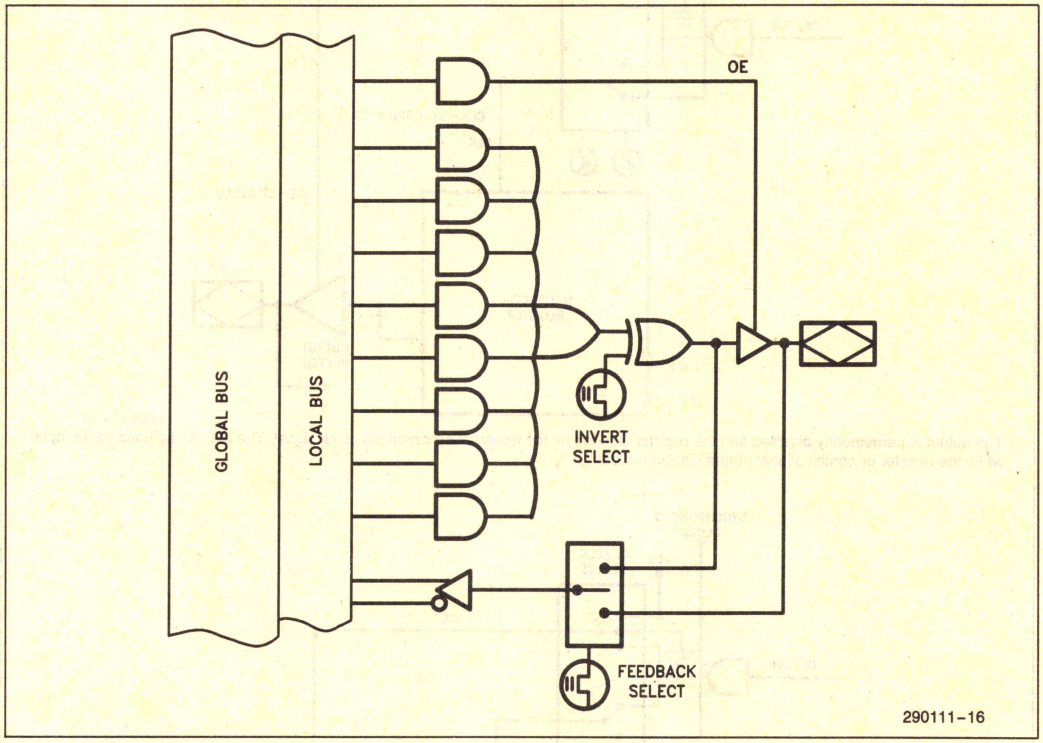
The output is permanently disabled and the register is clocked via the product term. This allows gated clocks that may be generated elsewhere in the 5C180. The pin can be used as in input while the register or combinational output can be fed back.

Figure 8b. Global Macrocell Additional OE/CLK Selection

MACROCELL LOGIC + I/O CONFIGURATIONS

The 5C180 Input/Output Architecture provides each Macrocell with over 50 possible I/O configurations.

Figures 9 and 10 show the 5C180 basic I/O configurations for both the Local and Global Macrocells. Along with combinatorial, four register types are available. Each Macrocell may be independently programmed.

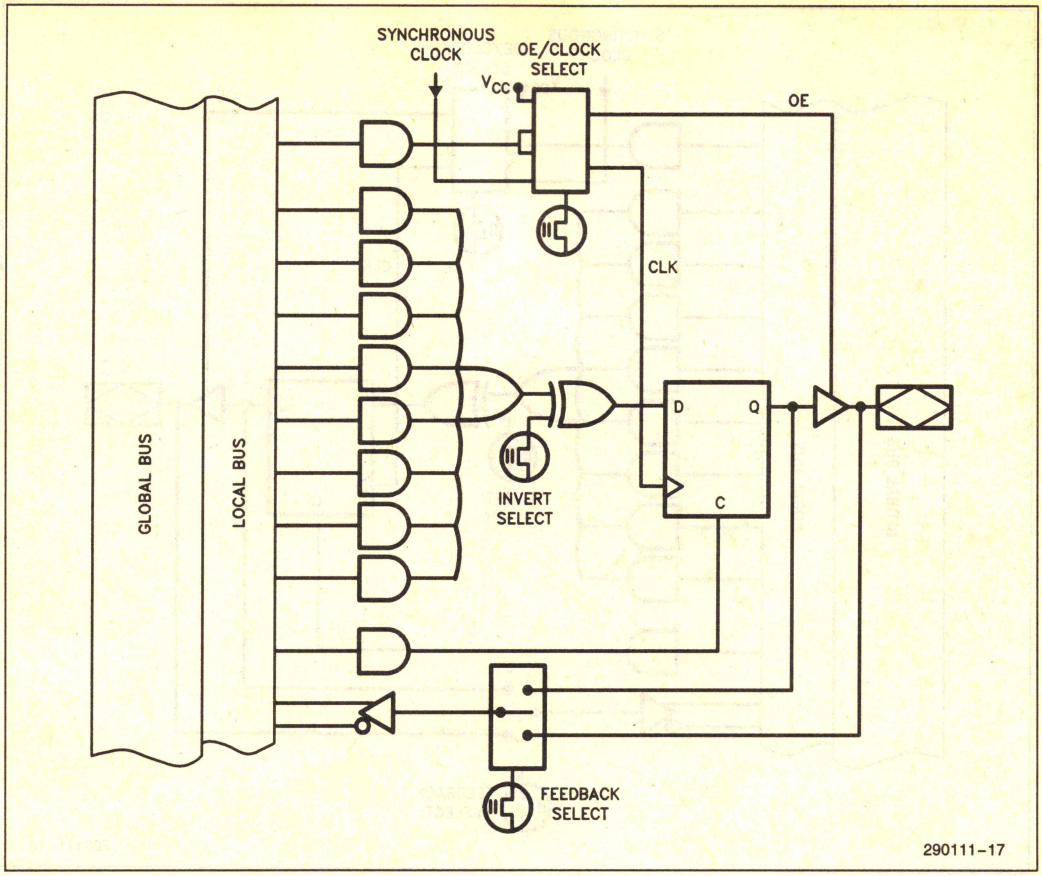


COMBINATORIAL

I/O Selection

Output/Polarity	Feedback	Bus
Combinatorial/High	Comb, Pin, None	Local
Combinatorial/Low	Comb, Pin, None	Local
None	Comb	Local
None	Pin	Local

Figure 9. Local Macrocell I/O Configurations



D-TYPE FLIP-FLOP

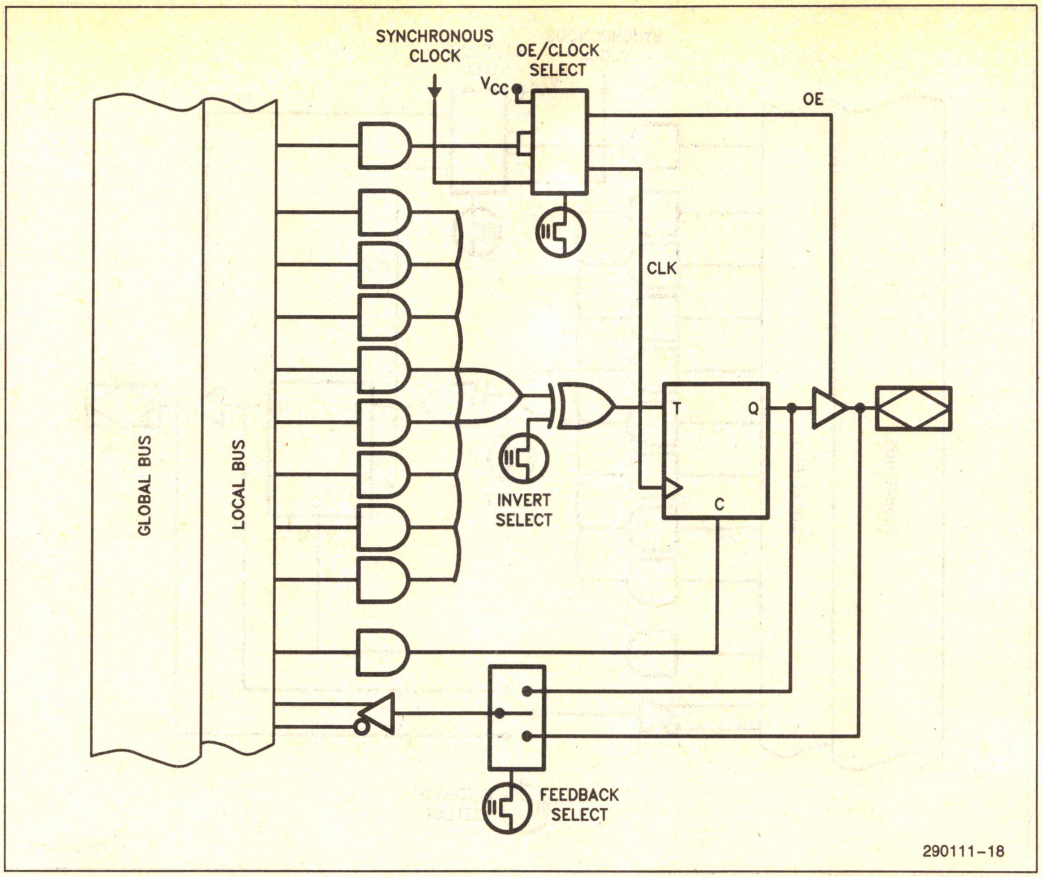
I/O Selection

Output/Polarity	Feedback	Bus
D-Register/High	D-Register, Pin, None	Local
D-Register/Low	D-Register, Pin, None	Local
None	D-Register	Local
None	Pin	Local

Function Table

D	Q_n	Q_{n+1}
0	0	0
0	1	0
1	0	1
1	1	1

Figure 9. Local Macrocell I/O Configurations (Continued)



TOGGLE FLIP-FLOP

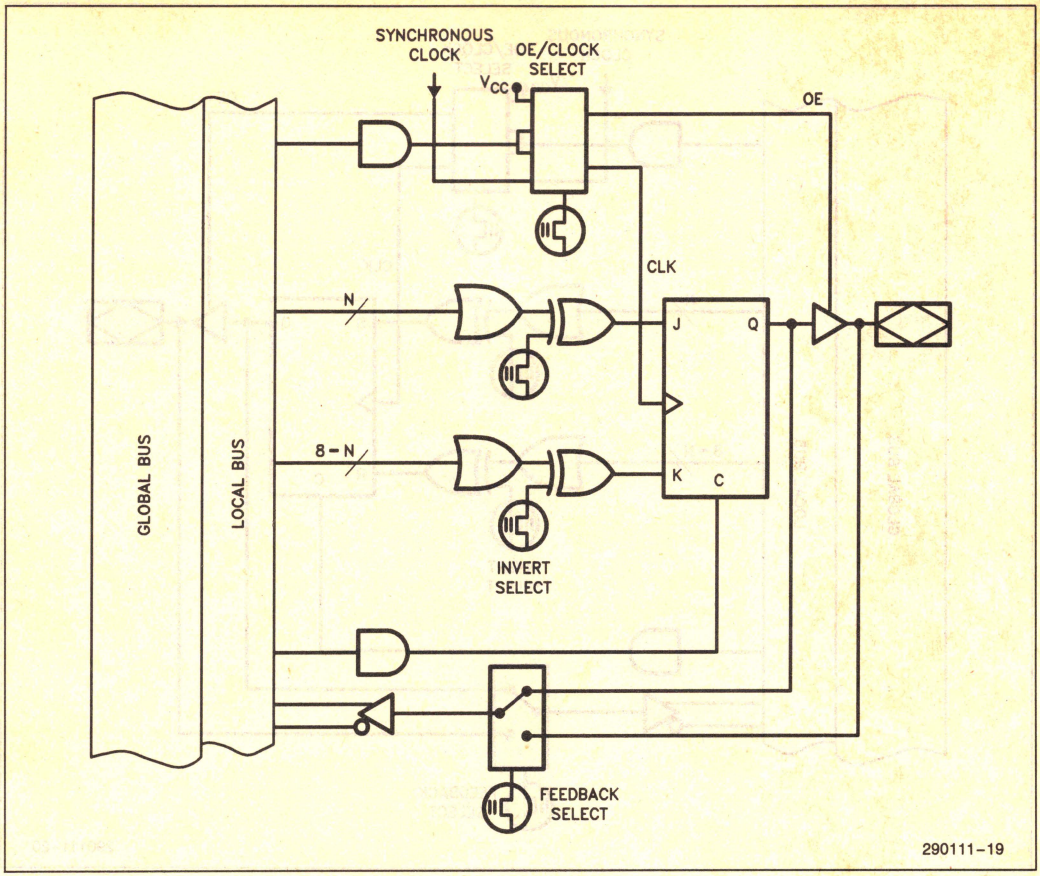
I/O Selection

Output/Polarity	Feedback	Bus
T-Register/High	T-Register, Pin, None	Local
T-Register/Low	T-Register, Pin, None	Local
None	T-Register	Local
None	Pin	Local

Function Table

T	Q_n	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

Figure 9. Local Macrocell I/O Configurations (Continued)



JK FLIP-FLOP

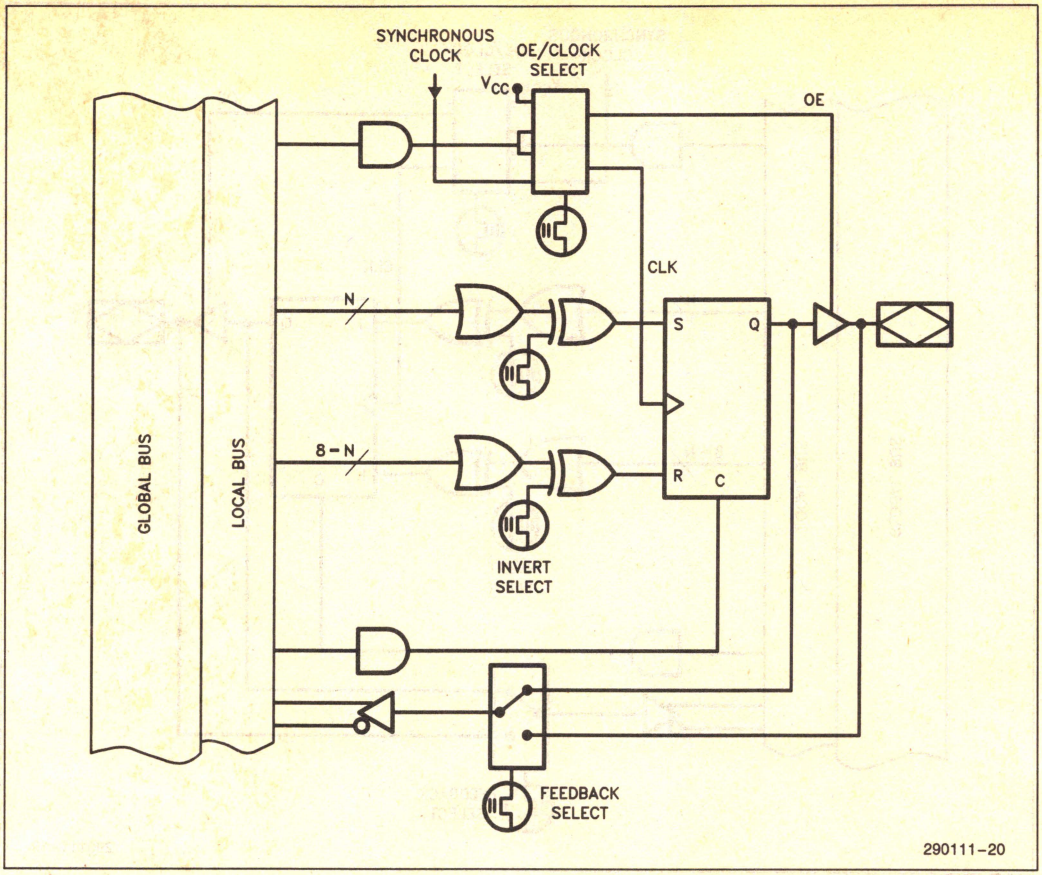
I/O Selection

Output/Polarity	Feedback	Bus
JK Register/High	JK Register, None	Local
JK Register/Low	JK Register, None	Local
None	JK Register	Local

Function Table

J	K	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Figure 9. Local Macrocell I/O Configurations (Continued)



SR FLIP-FLOP

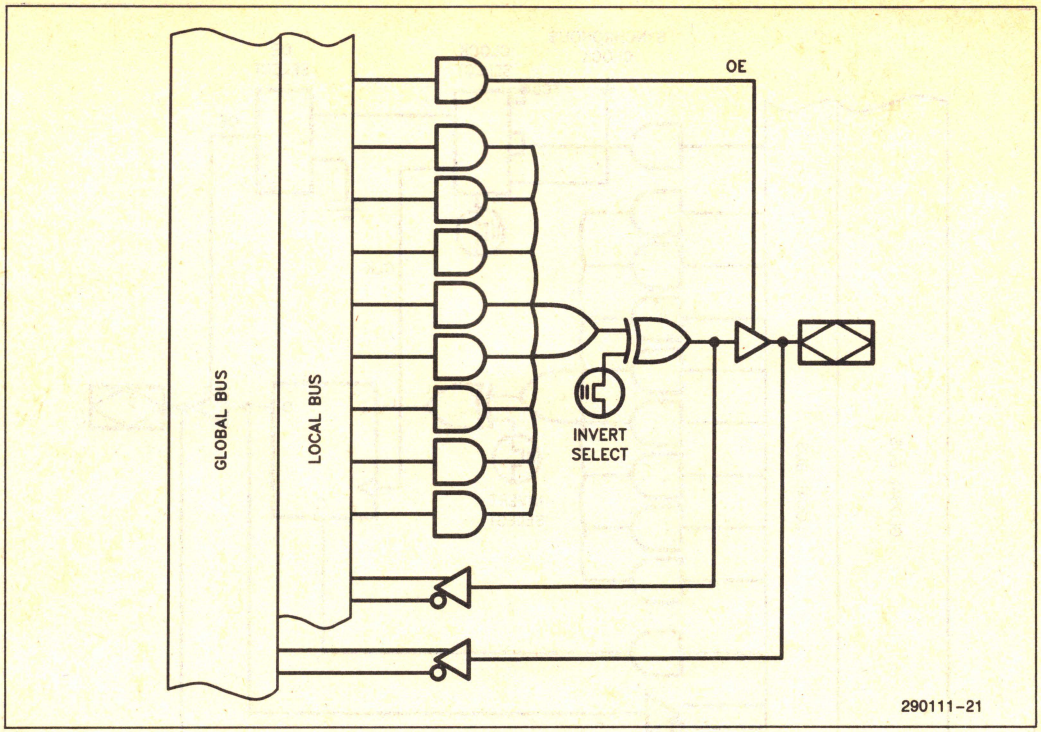
I/O Selection

Output/Polarity	Feedback	Bus
SR Register/High	SR Register, None	Local
SR Register/Low	SR Register, None	Local
None	SR Register	Local

Function Table

S	R	Q_n	$Q_n + 1$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1

Figure 9. Local Macrocell I/O Configurations (Continued)

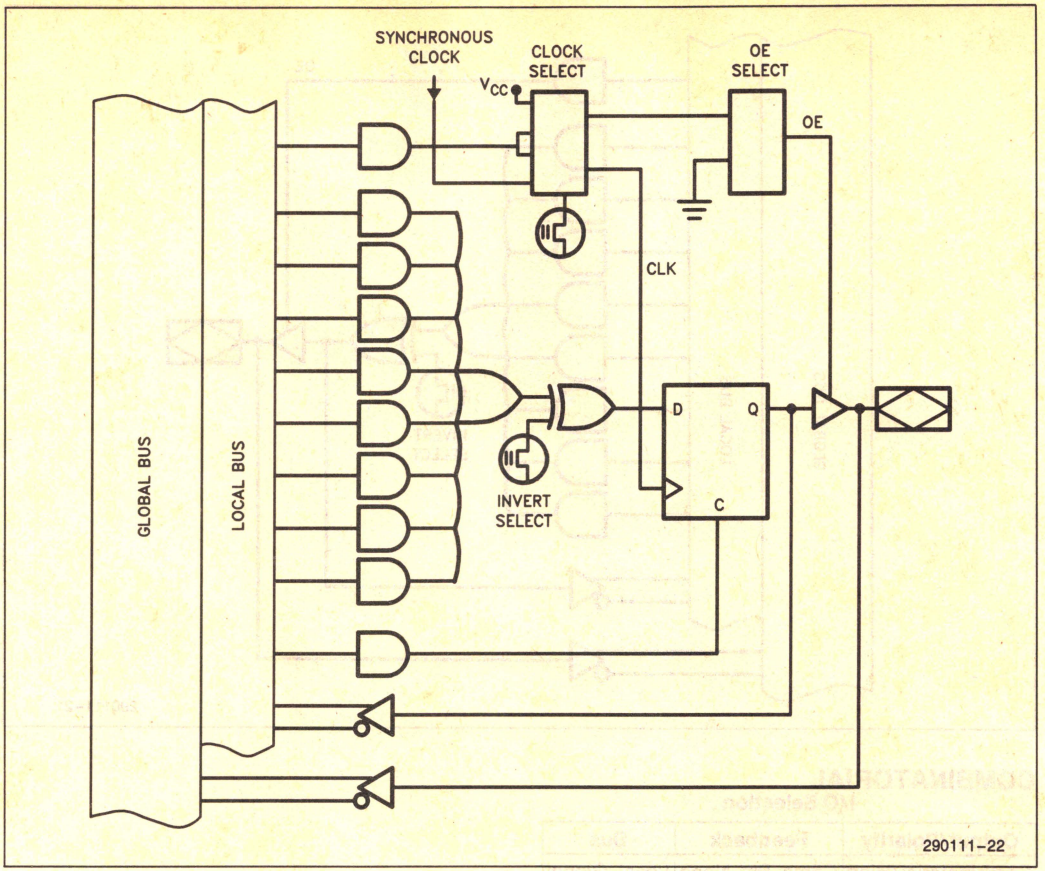


COMBINATORIAL

I/O Selection

Output/Polarity	Feedback	Bus
Combinatorial/High	Comb, Pin, None	Local, Global
Combinatorial/Low	Comb, Pin, None	Local, Global
None	Comb	Local, Global
None	Pin	Global
None	Comb/Pin	Local/Global

Figure 10. Global Macrocell I/O Configurations



D-TYPE FLIP-FLOP

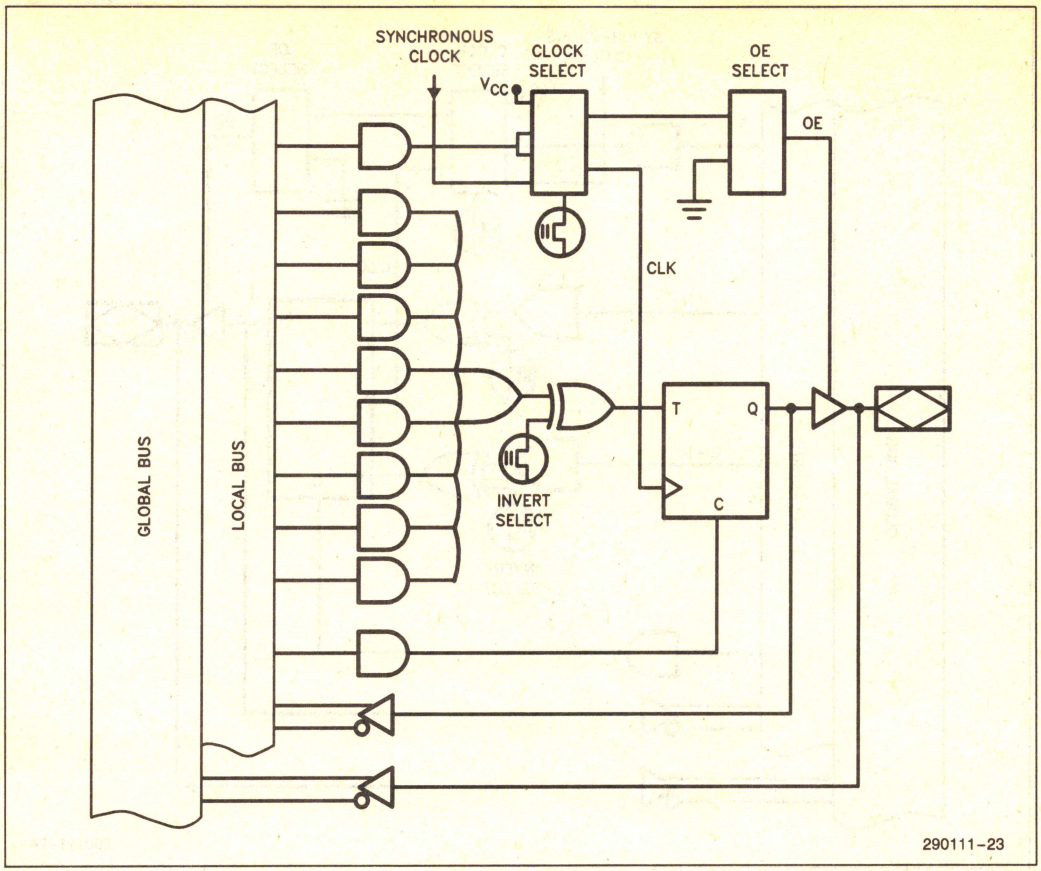
I/O Selection

Output/Polarity	Feedback	Bus
D-Register/High	D-Register, Pin, None	Local, Global
D-Register/Low	D-Register, Pin, None	Local, Global
None	D-Register	Local, Global
None	Pin	Global
None	D-Register/Pin	Local/Global

Function Table

D	Q_n	Q_{n+1}
0	0	0
0	1	0
1	0	1
1	1	1

Figure 10. Global Macrocell I/O Configurations (Continued)



TOGGLE FLIP-FLOP

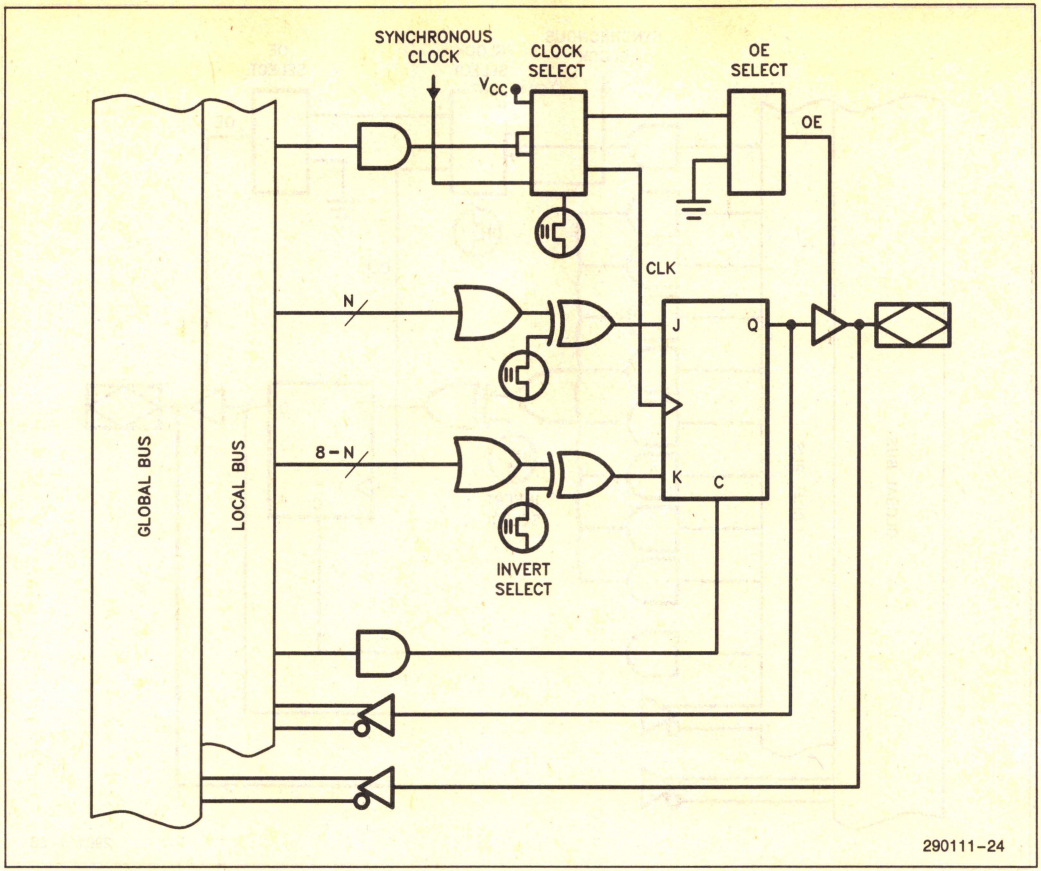
I/O Selection

Output/Polarity	Feedback	Bus
T-Register/High	T-Register, Pin, None	Local, Global
T-Register/Low	T-Register, Pin, None	Local, Global
None	T-Register	Local, Global
None	Pin	Global
None	T-Register/Pin	Local/Global

Function Table

T	Q_n	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

Figure 10. Global Macrocell I/O Configurations (Continued)



JK FLIP-FLOP

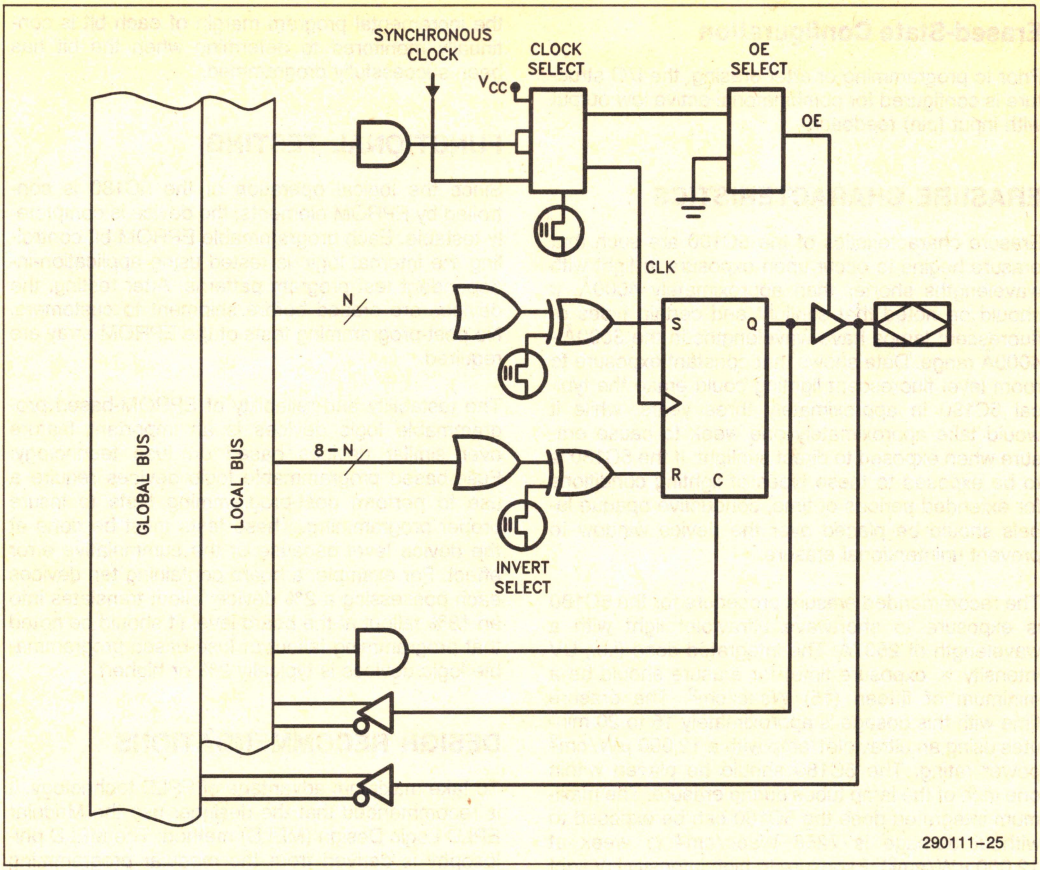
I/O Selection

Output/Polarity	Feedback	Bus
JK Register/High	JK Register, None	Local, Global
JK Register/Low	JK Register, None	Local, Global
None	JK Register	Local
None	JK Register/Pin	Local/Global

Function Table

J	K	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Figure 10. Global Macrocell I/O Configurations (Continued)



SR FLIP-FLOP

I/O Selection

Output/Polarity	Feedback	Bus
SR Register/High	SR Register, None	Local, Global
SR Register/Low	SR Register, None	Local, Global
None	SR Register	Local
None	SR Register/Pin	Local/Global

Function Table

S	R	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1

Figure 10. Global Macrocell I/O Configurations (Continued)

Erased-State Configuration

Prior to programming or after erasing, the I/O structure is configured for combinatorial active low output with input (pin) feedback.

ERASURE CHARACTERISTICS

Erasure characteristics of the 5C180 are such that erasure begins to occur upon exposure to light with wavelengths shorter than approximately 4000Å. It should be noted that sunlight and certain types of fluorescent lamps have wavelengths in the 3000Å–4000Å range. Data shows that constant exposure to room level fluorescent lighting could erase the typical 5C180 in approximately three years, while it would take approximately one week to cause erasure when exposed to direct sunlight. If the 5C180 is to be exposed to these types of lighting conditions for extended periods of time, conductive opaque labels should be placed over the device window to prevent unintentional erasure.

The recommended erasure procedure for the 5C180 is exposure to shortwave ultraviolet light with a wavelength of 2537Å. The integrated dose (i.e., UV intensity $\times$ exposure time) for erasure should be a minimum of fifteen (15) Wsec/cm^2. The erasure time with this dosage is approximately 15 to 20 minutes using an ultraviolet lamp with a 12,000 μW/cm^2 power rating. The 5C180 should be placed within one inch of the lamp tubes during erasure. The maximum integrated dose the 5C180 can be exposed to without damage is 7258 Wsec/cm^2 (1 week at 12,000 μW/cm^2). Exposure to high intensity UV light for longer periods may cause permanent damage to the device.

PROGRAMMING CHARACTERISTICS

Initially, and after erasure, all the EPROM control bits of the 5C180 are connected. Each of the connected control bits are selectively disconnected by programming the EPROM cells into their "on" state. Programming voltage and waveform specifications are available by request from Intel to support programming of the 5C180.

int$_e$ligent Programming™ Algorithm

The 5C180 supports the int$_e$ligent Programming Algorithm which rapidly programs Intel H-ELPDs (and EPROMs) using an efficient and reliable method. The int$_e$ligent Programming Algorithm is particularly suited to the production programming environment. This method greatly decreases the overall programming time while programming reliability is ensured as the incremental program margin of each bit is continually monitored to determine when the bit has been successfully programmed.

FUNCTIONAL TESTING

Since the logical operation of the 5C180 is controlled by EPROM elements, the device is completely testable. Each programmable EPROM bit controlling the internal logic is tested using application-independent test program patterns. After testing, the devices are erased before shipment to customers. No post-programming tests of the EPROM array are required.

The testability and reliability of EPROM-based programmable logic devices is an important feature over similar devices based on fuse technology. Fuse-based programmable logic devices require a use to perform post-programming tests to insure proper programming. These tests must be done at the device level because of the cummulative error effect. For example, a board containing ten devices each possessing a 2% device fallout translates into an 18% fallout at the board level (it should be noted that programming fallout of fuse-based programmable logic devices is typically 2% or higher).

DESIGN RECOMMENDATIONS

To take maximum advantage of EPLD technology, it is recommended that the designer use the Modular EPLD Logic Design (MELD) method. The MELD philosophy is derived from the modular programming method used in software development. In a modular software development environment, the engineer designs a modular program (typically on a development system), stores it in memory (EPROM), and tests the module for functionality. A hardware designer using EPLDs can use this same approach when designing logic. The designer develops a modular logic design on the Intel Programmable Logic Development System (iPLDS), stores it in "memory" (the EPROM control elements of the EPLD), and again tests the module for functionality. If the design is in error, the logic designer reprograms the EPLD with his new design as easily as a software designer can download a new program into memory.

The MELD philosophy is new to programmable logic because EPROM-based PLDs are new. A modular logic development process using fused-based PLDs would be wasteful since a fuse-based device cannot be erased and re-used.

For proper operation, it is recommended that all input and output pins be constrained to the voltage range GND < (V_{IN} or V_{OUT}) < V_{CC}. Unused inputs should be tied to an appropriate logic level (e.g., either V_{CC} or GND) to minimize device power consumption. Reserved pins (as indicated in the iPLS II REPORT file) should be left floating (no connect) so that the pin can attain the appropriate logic level. A power supply decoupling capacitor of at least 0.2 μf must be connected directly between V_{CC} and GND.

DESIGN SECURITY

A single EPROM bit provides a programmable design security feature that controls the access to the data programmed into the device. If this bit is set, a proprietary design within the device cannot be copied. This EPROM security bit enables a higher degree of design security than fused-based devices since programmed data within EPROM cells is invisible even to microscopic evaluation. The EPROM security bit, along with all the other EPROM control bits, will be reset by erasing the device.

LATCH-UP IMMUNITY

All of the input, I/O, and clock pins of the 5C180 have been designed to resist latch-up which is inherent in inferior CMOS structures. The 5C180 is designed with Intel's proprietary CHMOS II-E EPROM process. Thus, each of the 5C180 pins will not experience latch-up with currents up to 100 mA and voltages ranging fronm -1V to $V_{CC}+1$V. Furthermore, the programming pin is designed to resist latch-up to the 13.5V maximum device limit.

INTEL PROGRAMMABLE LOGIC DEVELOPMENT SYSTEM II (iPLDS II)

The iPLDS II graphically shown in Figure 11 provides all the tools needed to design with Intel H-Series EPLDs or compatible devices. In addition to providing development assistance, iPLDS II insulates the user from having to know all the intricate details of EPLD architecture (the machine will optimize a design to benefit from architectural features). It contains comprehensive third generation software that supports four different design entry methods, minimizes logic, does automatic pin assignments and produces the best design fit for the selected EPLD. It is user friendly with guided menus, on-line Help messages and soft key inputs.

In addition, the iPLDS II contains programmer hardware in the form of an iUP-PC Universal Programmer-Personal Computer to enable the user to program EPLDs, read and verify programmed devices and also to graphically edit programming files. The software generates industry standard JEDEC object code output files which can be downloaded to other programmers as well.

iPLDS II has interfaces to popular schematic capture packages (including Dash series from Future-NET* and PC-CAPS** from P-CAD) to enable designs to be entered using schematics. A more integrated schematic entry method is provided by SCHEMA II-PLD, a low-cost schematic capture package that supports EPLD primitives and user-defined macro symbols. SCHEMA II-PLD contains the EPLD Design Manager, which provides a single user interface to both SCHEMA II-PLD and iPLS II software. The other design formats supported are Boolean equation entry and State Machine design entry.

The iPLDS II operates on the IBM† PC/XT, PC/AT, or other compatible machine with the following configuration:

1. At least one floppy disk drive and hard disk drive.
2. MS-DOS‡ Operating System Version 3.0 or greater.
3. 640K Memory.
4. Intel iUP-PC Universal Programmer-Personal Computer (supplied with iPLDS II).
5. GUPI LOGIC Adaptor
6. A color monitor is suggested.

Detailed information on the Intel Programmable Logic Development System II is contained in a separate Intel data sheet. (Order Number: 280168)

* FutureNET is a registered trademark of FutureNET Corporation. DASH is a trademark of FutureNET Corporation.

** PC-CAPS is a trademark of P-CAD Corporation.

† IBM Personal Computer is a registered trademark of International Business Machines Corporation.

‡ MS-DOS is a registered trademark of Microsoft Corporation.

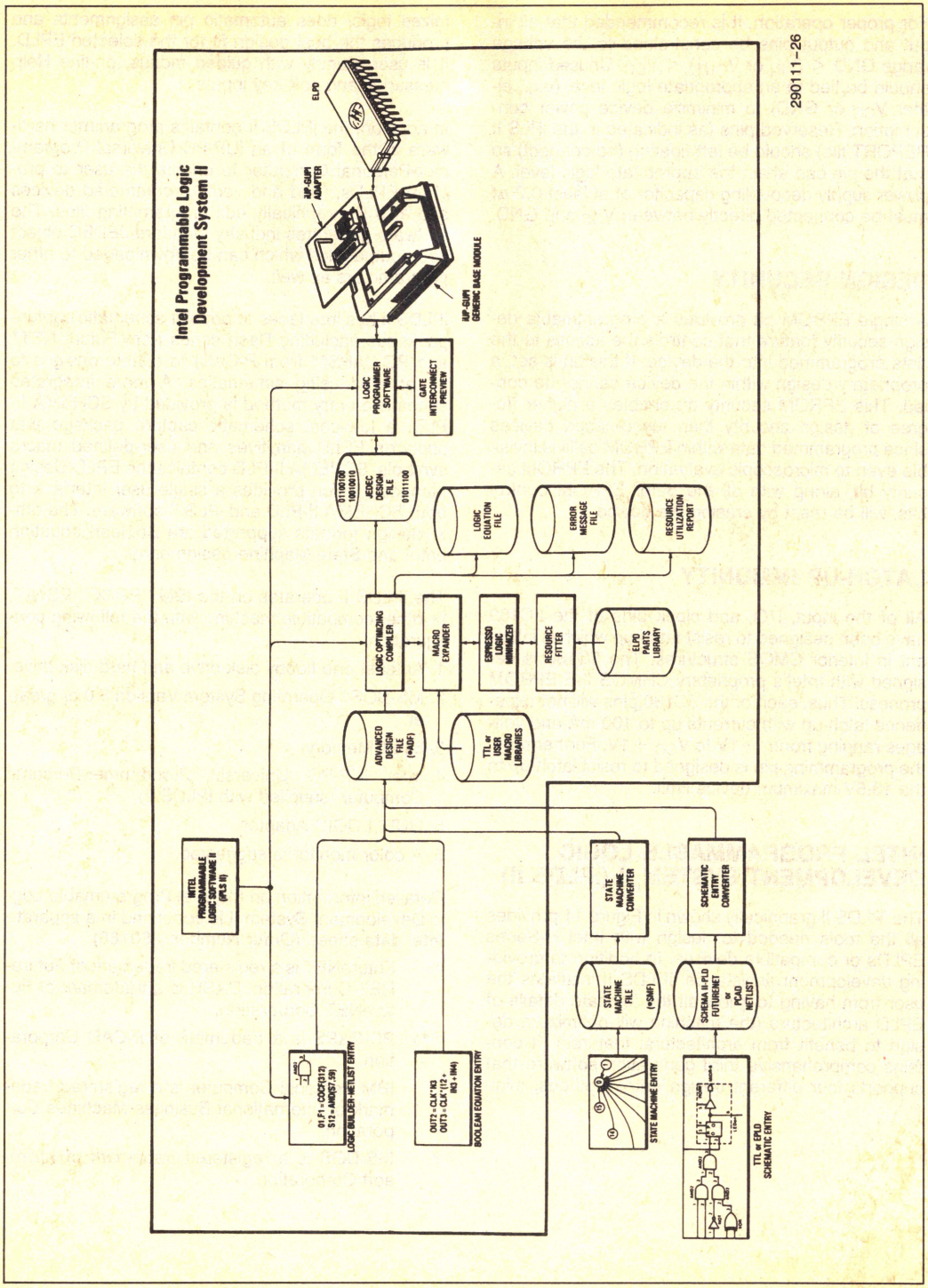


Figure 11. iPLDS II Intel Programmable Logic Development System

ABSOLUTE MAXIMUM RATINGS*

Symbol	Parameter	Min	Max	Units
V_{CC}	Supply Voltage(1)	−2.0	7.0	V
V_{PP}	Programming Supply Voltage(1)	−2.0	13.5	V
V_I	DC Input Voltage(1)(2)	−0.5	V_{CC}+0.5	V
t_{stg}	Storage Temperature	−65	+150	°C
t_{amb}	Ambient Temperature(3)	−10	+85	°C

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

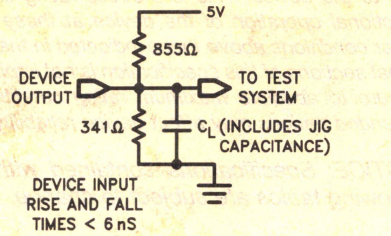
NOTES:
1. Voltages with respect to ground.
2. Minimum DC input is −0.5V. During transitions, the inputs may undershoot to −2.0V for periods less than 20 ns under no load conditions.
3. Under bias. Extended temperature versions are also available.

D.C. CHARACTERISTICS T_A = 0° to +70°C, V_{CC} = 5V ±5%

Symbol	Parameter/Test Conditions	Min	Typ	Max	Unit
V_{IH}(4)	High Level Input Voltage	2.0		V_{CC} + 0.3	V
V_{IL}(4)	Low Level Input Voltage	−0.3		0.8	V
V_{OH}(5)	High Level Output Voltage I_O = −4.0 mA D.C., V_{CC} = min.	2.4			V
V_{OL}	Low Level Output Voltage I_O = 4.0 mA D.C., V_{CC} = min.			0.45	V
I_I	Input Leakage Current V_{CC} = max., GND < V_{OUT} < V_{CC}			±10	µA
I_{OZ}	Output Leakage Current V_{CC} = max., GND < V_{OUT} < V_{CC}			±10	µA
I_{SC}(6)	Output Short Circuit Current V_{CC} = max., V_{OUT} = 0.5V				mA
I_{SB}(7)	Standby Current V_{CC} = max., V_{IN} = V_{CC} or GND, Standby mode		20	150	µA
I_{CC}	Power Supply Current V_{CC} = max., V_{IN} = V_{CC} or GND, No load, Input Freq. = 1 MHz Active mode (Turbo = Off), Device prog. as 4 12-bit Ctr.		30	45	mA

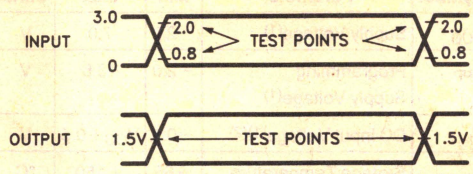
NOTES:
4. Absolute values with respect to device GND; all over and undershoots due to system or tester noise are included.
5. I_O at CMOS levels (3.84 V) = −2 mA
6. Not more than 1 output should be tested at a time. Duration of that test must not exceed 1 second.
7. With Turbo Bit Off, device automatically enters standby mode approximately 100 ns after last input transition.

A.C. TESTING LOAD CIRCUIT



C_L = 50 pF

A.C. TESTING INPUT, OUTPUT WAVEFORM



A.C. Testing: Inputs are Driven at 3.0V for a Logic "1" and 0V for a Logic "0". Timing Measurements are made at 2.0V for a Logic "1" and 0.8V for a Logic "0" on inputs. Outputs are measured at a 1.5V point.

A.C. CHARACTERISTICS T_A = 0°C to +70°C, V_{CC} = 5V ±5%, Turbo Bit On[8]

Symbol	From	To	5C180-75			5C180-90			Non[10] Turbo Adjust	Unit
			Min	Typ	Max	Min	Typ	Max		
t_{PD1}	Input	Comb. Output			70			85	+30	ns
t_{PD2}	Local I/O	Comb. Output			75			90	+30	ns
t_{PDG}	Global I/O	Comb. Output			70			85	+30	ns
t_{PZX}[9]	I or I/O	Output Enable			75			90	+30	ns
t_{PXZ}[9]	I or I/O	Output Disable			75			90	+30	ns
t_{CLR}	Asynch. Reset	Q Reset			75			90	+30	ns

NOTES:
8. Typ. Values are at T_A = 25°C, V_{CC} = 5V, Active Mode.
9. t_{PZX} and t_{PXZ} are measured at ±0.5V from steady state voltage as driven by spec. output load. t_{PXZ} is measured with C_L = 5 pF.
10. If device is operated with Turbo Bit Off (Non-Turbo Mode), increase time by amount shown.

CAPACITANCE

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
C_{IN}	Input Capacitance	V_{IN} = 0V, f = 1.0 MHz			15	pF
C_{OUT}	Output Capacitance	V_{OUT} = 0V, f = 1.0 MHz			15	pF
C_{CLK}	Clock Pin Capacitance	V_{OUT} = 0V, f = 1.0 MHz			25	pF
C_{VPP}	V_{PP} Pin Capacitance	Pin 19, V_{OUT} = 0V, f = 1.0 MHz			160	pF

SYNCHRONOUS CLOCK MODE A.C. CHARACTERISTICS

T_A = 0°C to +70°C, V_{CC} = 5V ±5%, Turbo Bit On[11]

Symbol	Parameter	5C180-75			5C180-90			Non[12] Turbo Adjust	Unit
		Min	Typ	Max	Min	Typ	Max		
f_{MAX}	Max Frequency $1/t_{SU}$–No Feedback	19.6			16.1				MHz
f_{CNT}	Max. Count Frequency $1/t_{CNT}$–With Feedback	15.1			12.2				MHz
t_{SU1}	Input Setup Time to Clk	51			62			+30	ns
t_{SU2}	Local I/O Setup Time to Clk	56			67			+30	ns
t_{SUG}	Global I/O Setup Time to Clk	51			62			+30	ns
t_H	I or I/O Hold after Clk High	0			0				ns
t_{CO}	Clk High to Output Valid			30			35		ns
t_{CNT}	Register Output Feedback to Register Input–Internal Path			66			82	+30	ns
t_{CH}	Clk High Time	25			30				ns
t_{CL}	Clk Low Time	25			30				ns

NOTES:
11. Typ. Values are at T_A = 25°C, V_{CC} = 5V, Active Mode.
12. If device is operated with Turbo Bit Off (Non-Turbo Mode), increase time by amount shown.

ASYNCHRONOUS CLOCK MODE A.C. CHARACTERISTICS

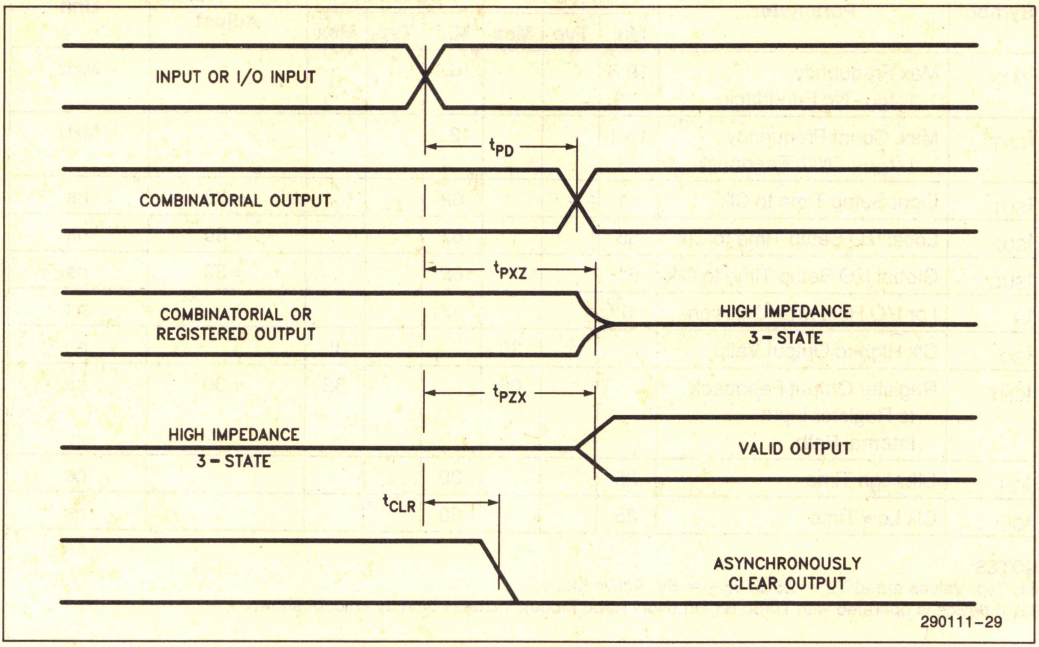
T_A = 0°C to +70°C, V_{CC} = 5V ±5%, Turbo Bit On[13]

Symbol	Parameter	5C180-75			5C180-90			Non[14] Turbo Adjust	Unit
		Min	Typ	Max	Min	Typ	Max		
f_{AMAX}	Max. Frequency $1/t_{ASU}$–No Feedback	66.7			40.0				MHz
f_{ACNT}	Max. Frequency $1/t_{ACNT}$–With Feedback	15.1			12.2				MHz
t_{ASU1}	Input Setup Time to Asynch. Clock	17			23			+30	ns
t_{ASU2}	I/O Setup Time to Asynch. Clock	22			28			+30	ns
t_{AH}	Input or I/O Hold to Asynch. Clock	30			30				ns
t_{ACO}	Asynch. Clk to Output Valid			75			90		ns
t_{ACNT}	Register Output Feedback to Register Input–Internal Path			66			82	+30	ns
t_{ACH}	Asynch. Clk High Time	25			30				ns
t_{ACL}	Asynch. Clk Low Time	25			30				ns

NOTES:
13. Typ. Values are at T_A = 25°C, V_{CC} = 5V, Active Mode.
14. If device is operated with Turbo Bit Off (Non-Turbo Mode), increase time by amount shown.

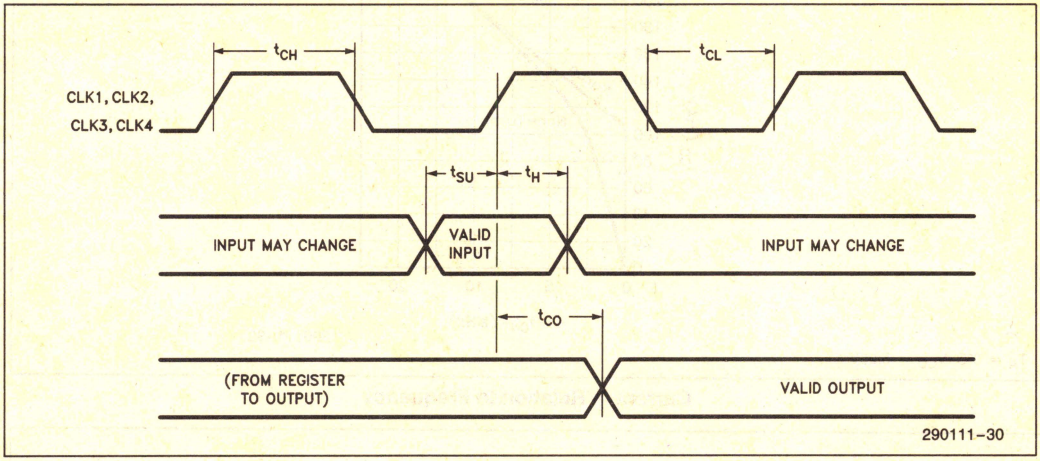
SWITCHING WAVEFORMS

COMBINATORIAL MODE

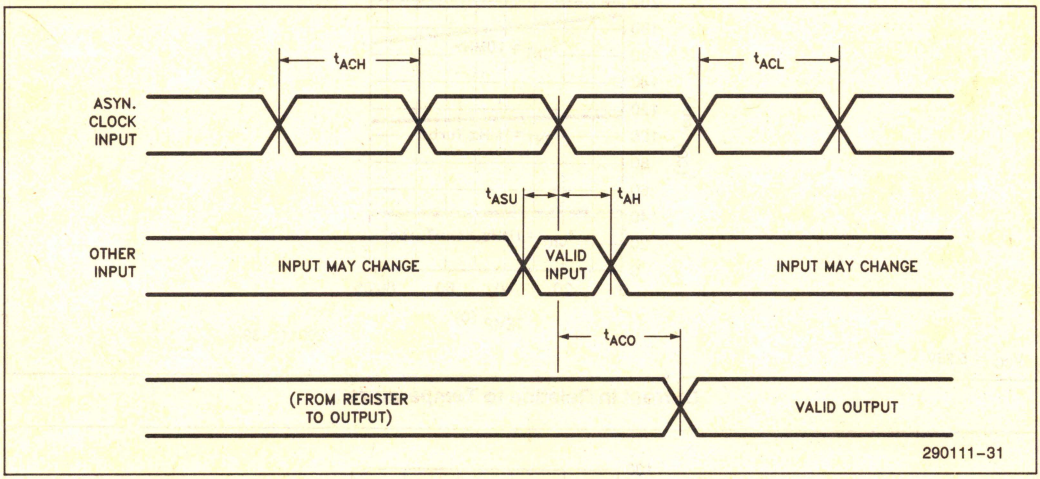


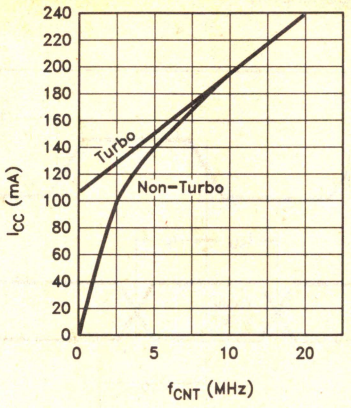
SWITCHING WAVEFORMS (Continued)

SYNCHRONOUS CLOCK MODE



ASYNCHRONOUS CLOCK MODE

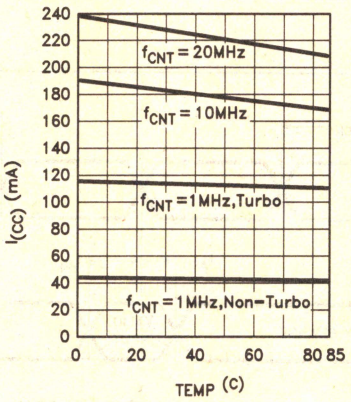




290111-32

T_A = 0°C, V_{CC} = 5.25V

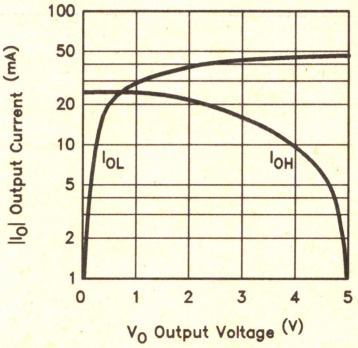
Current in Relation to Frequency



290111-33

V_{CC} = 5.25V

Current in Relation to Temperature



290111-34

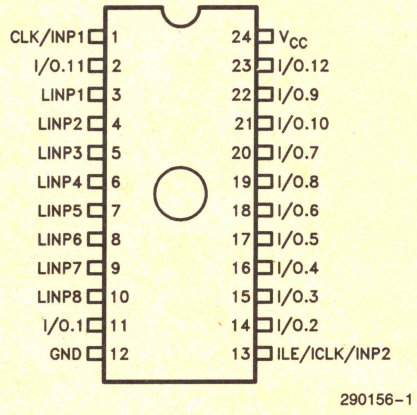
Output Drive Current in Relation to Voltage

5AC312
ERASABLE PROGRAMMABLE LOGIC DEVICE

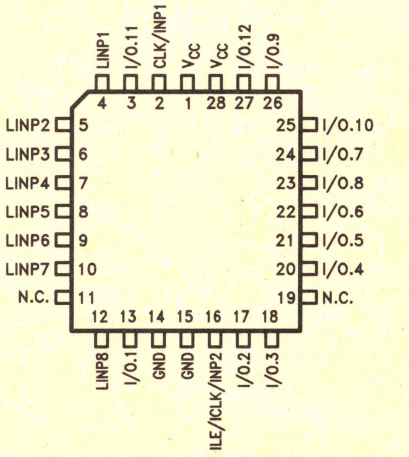
- High Performance LSI Semi-Custom Logic Alternative for Low-End Gate Arrays, TTL, and 74HC- or 74HCT SSI and MSI Logic
- High Speed tpd (max) 25 ns, 40 MHz Operating Performance
- Erasable Array for 100% Generic Testability
- CHMOS III-E EPROM Technology based; UV-Erasable
- Low Power; 150 μA Standby Current
- Programmable Security Bit Allows 100% Protection of Proprietary Designs

5AC312 KEY FEATURES

- 12 Macrocells with Programmable I/O Architecture; Up To 22 Inputs (10 Dedicated, 12 I/O) or 12 Outputs
- 8 Programmable Inputs; Each Can Be Programmed Individually to Implement Latch, Register or Flow-Through Structure; Synchronous or Asynchronous Operation
- Software-Supported Product Term Allocation between Adjacent Macrocells
- Dual Feedback on All Macrocells for Buried Register Implementation and Input Usage
- 2 Product Terms on All Macrocell Control Signals
- Programmable Power Option for "Stand-By" Operation
- Available in 24-Pin 0.3" DIP and 28-Pin PLCC Packages
  (See Packaging Spec., Order Number #231369)



290156-1



290156-2

Figure 1. Pin Configurations

November 1987
Order Number: 290156-001

INTRODUCTION

The Intel 5AC312 CHMOS EPLD (Erasable Programmable Logic Device) represents an innovative approach to overcoming the primary limitations of standard PLDs. Due to a proprietary I/O architecture and macrocell structure, the 5AC312 is capable of implementing high performance logic functions more effectively than previously possible. It can be used as an alternative to low-end gate arrays, multiple programmable logic devices or LS-, HC- or HCT SSI and MSI logic devices.

The 5AC312 uses advanced CHMOS EPROM cells as logic control elements instead of poly-silicon fuses. This technology allows the 5AC312 to operate at levels necessary in high performance systems while significantly reducing the power consumption of this device. Its programmable stand-by function reduces power consumption to almost "zero" in applications where speed is traded for power consumption.

ARCHITECTURE DESCRIPTION

The architecture of the 5AC312 is based on the familiar "Sum-Of-Products" programmable AND, fixed OR structure, though the 5AC312 macrocell contains a number of significant functional enhancements. This device can implement both combinational and sequential logic functions through a highly flexible macrocell and I/O structure. In addition, the 5AC312 has been designed to effectively implement both combinational-register and register-combinational-register forms of logic to easily accommodate state machine designs.

Figure 2 shows a global view of the 5AC312 architecture. The 5AC312 contains a total of 12 I/O macrocells, 8 user-programmable input structures, and 2 inputs that can be programmed to serve as either combinatorial inputs or clock inputs for the input and output register functions. Each macrocell is further sub-divided into 16 Product Terms with 8 Product Terms dedicated to the control signals OE, PRESET, ASYNCH. CLK and CLEAR, and 8 Product Terms available for the general data array.

The basic macrocell architecture of the 5AC312, shown in Figure 3, includes a user-programmable AND array and a user-configurable OR array. The inputs to the programmable AND array originate from the true and complement signals from the programmable input structure, the dedicated inputs, and the 24 feedback paths from the 12 I/O macrocells.

Programmable Input Structure

Figure 4 shows a block diagram of the 5AC312 input architecture. This device contains 8 user-program-

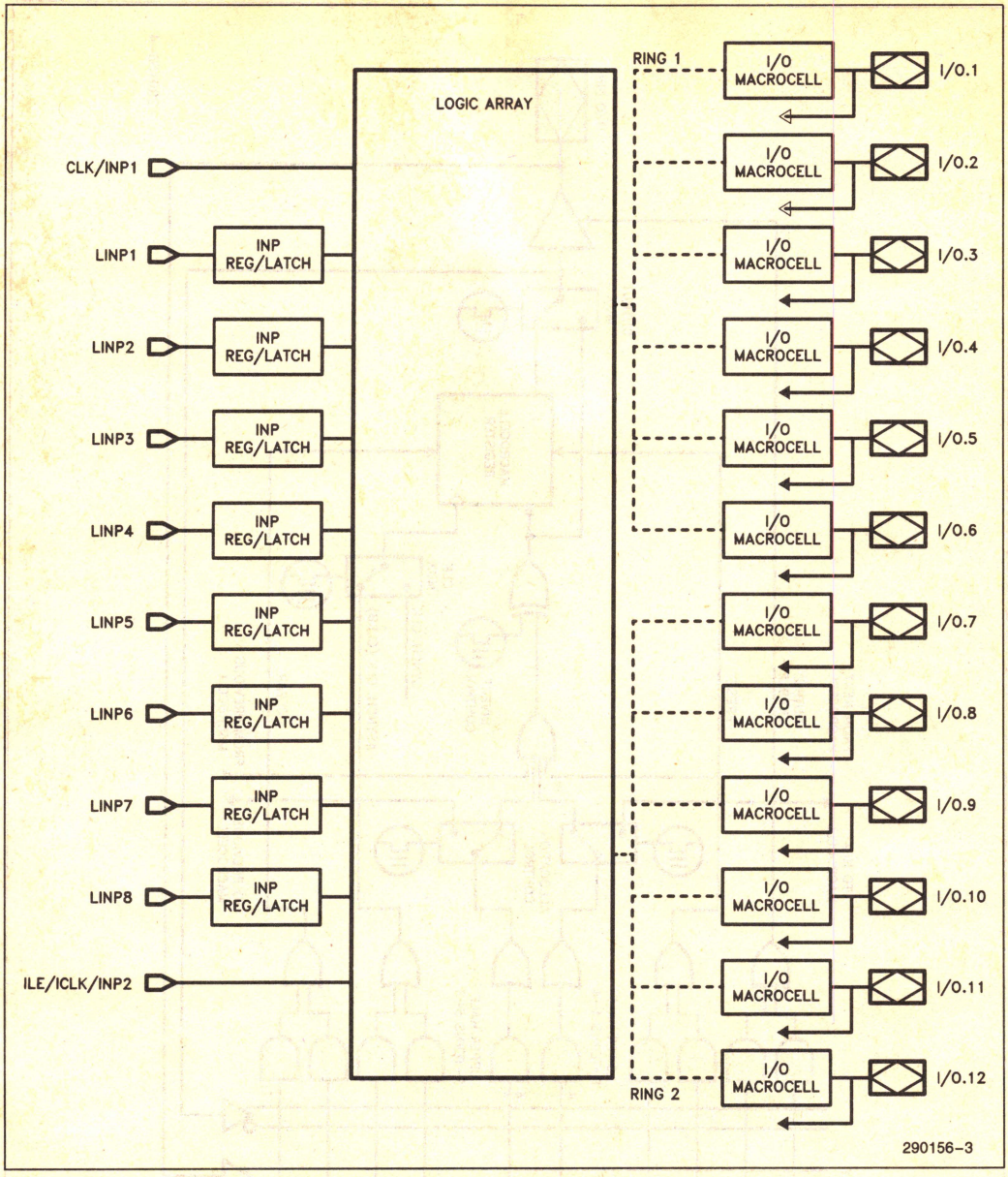


Figure 2. 5AC312 Architecture

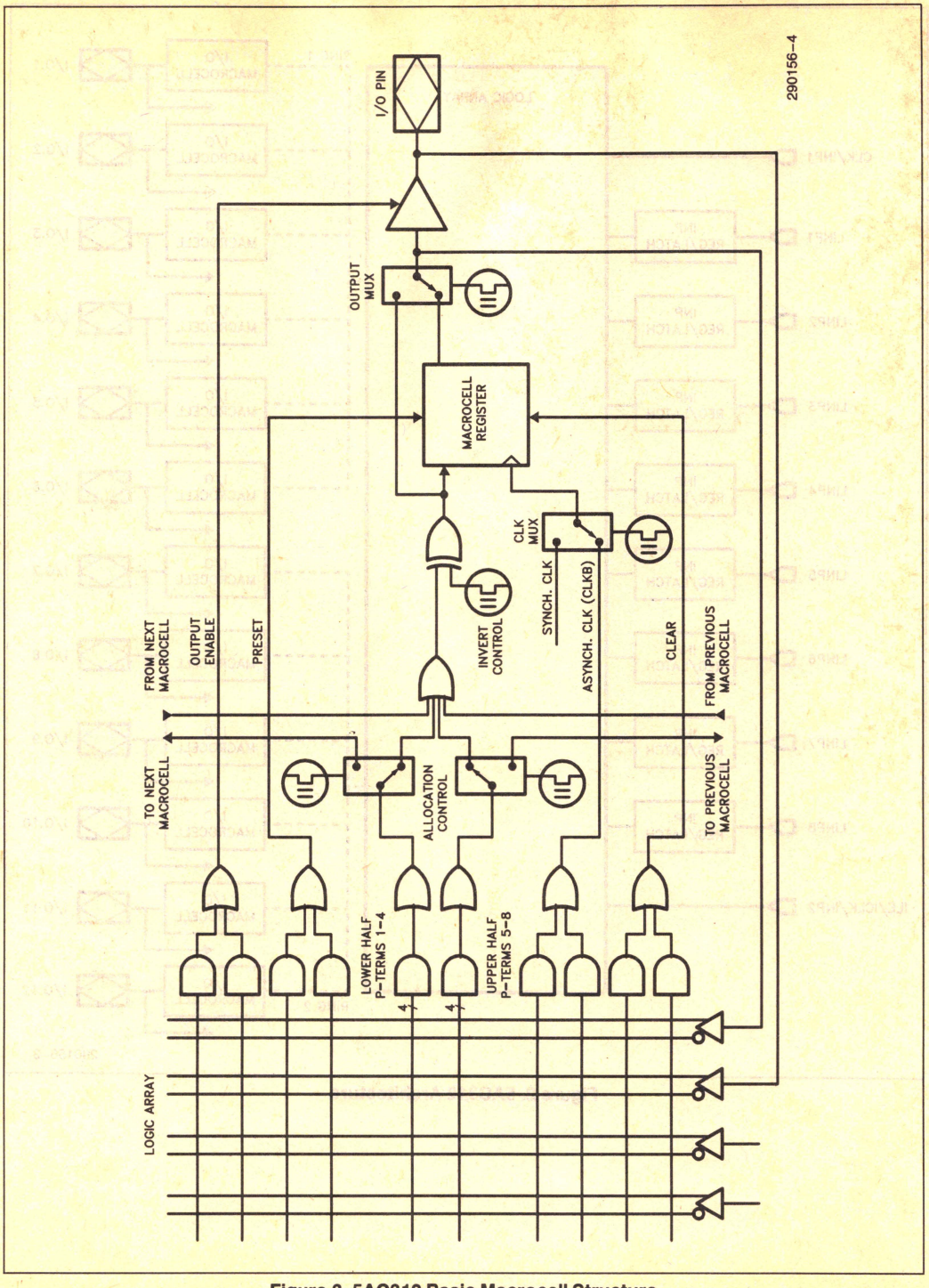
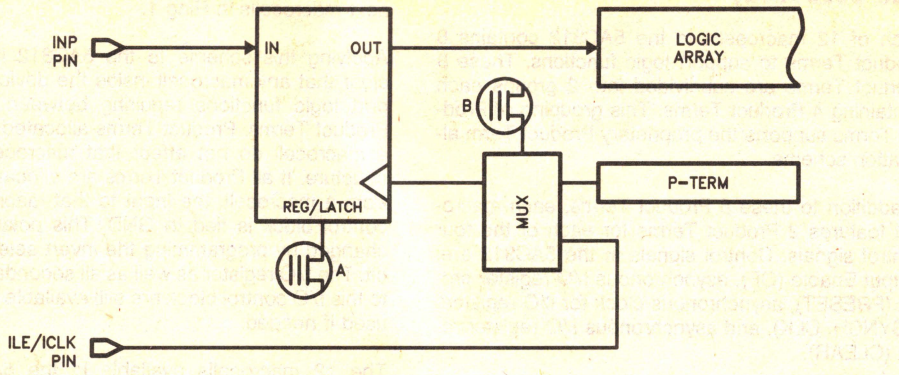


Figure 3. 5AC312 Basic Macrocell Structure



NOTE:
Flow-through input selected by connecting ILE P-Term to V_{CC}.

Figure 4. 5AC312 Input Structure

mable input structures that can be individually configured to work in one of five modes:

- — Input register (D-register), synchronous operation
- — Input register (D-register), asynchronous operation
- — Input latch (D-latch), synchronous operation
- — Input latch (D-latch), asynchronous operation
- — Flow-through input

The configuration is accomplished through the programming of EPROM architecture control bits by iPLS II V1.5 under user-control. If synchronous operation is chosen, pin 13 of the device becomes an ILE/ICLK (Input Latch Enable) input global to all input latch/register structures. For asynchronous operation, pin 13 can be used as a normal input (flow-through input) to the device while a separate Product Term in the control array is used to derive an ILE/ICLK signal for the input structure. Because the clock signal for each input structure can be individually selected, a mix between synchronously and asynchronously clocked input structures is also possible.

Table 1 shows the input latch/register function table with respect to the synchronous ILE/ICLK input.

Table 1. 5AC312 Input Latch/Register Functions

Input Type	ILE/ICLK	D	Q
Latch	H	H	H
Latch	H	L	L
Latch	L	X	Qn
D-FF	↓	H	H
D-FF	↓	L	L
Flow-Through	X	H	H
Flow-Through	X	L	L

H = HIGH Level L = LOW Level X = Don't Care

Macrocell Array

Each of 12 macrccells in the 5AC312 contains 8 Product Terms to support logic functions. These 8 Product Terms are subdivided into 2 groups each containing 4 Product Terms. This grouping of Product Terms supports the proprietary Product Term allocation scheme.

In addition to these 8 Product Terms, each macrocell features 2 Product Terms for each of the four control signals. Control signals in the 5AC312 are: Output Enable (OE), asynchronous I/O register preset (PRESET), asynchronous clock for I/O registers (ASYNCH. CLK), and asynchronous I/O register reset (CLEAR).

Product Term Allocation

Product Term allocation is defined as taking logic resources (p-terms) away from macrocells where they are not used to support demand for more than 8 Product Terms in other areas of the chip. In the 5AC312, this allocation can occur in increments of 4 p-terms between adjacent macrocells.

Example:

The logic function in macrocell 4 requires 16 p-terms. In this case, the iPLS II software allocates 4 p-terms from the previous macrocell in Ring 1 (macrocell 3) and 4 p-terms from the next macrocell in Ring 2 (macrocell 5) to accumulate a total of 16 p-terms (8 + 4 + 4). This implementation leaves macrocells 3 and 5 with a remainder of 4 p-terms each. These remaining p-terms in macrocells 3 and 5 can also be allocated away to or can be supplemented with p-terms from their respective previous/next macrocells in Ring 1.

Applying this scheme to the 5AC312 it becomes clear that any macrocell inside the device can support logic functions requiring between 0 and 16 Product Terms. Product Terms allocated away from a macrocell do not affect that macrocell's output structure. If all Product Terms are allocated "away" from a macrocell, the input to that macrocell's I/O control block is tied to GND. This polarity can be changed by programming the invert select EPROM bit. The I/O register as well as all secondary controls to this I/O control block are still available and can be used if needed.

The 12 macrocells available in the 5AC312 are grouped into two "rings" with 6 macrocells per ring. Product Terms can be allocated in a "shift register" mode inside a ring; allocation of Product Terms between the rings is not supported. The two rings are shown in Figure 2 and listed in Table 2.

The Product Term allocation scheme described above is automatically supported by iPLDS II V1.5 and is transparent to the user. Users can still use explicit pin assignments, but should assign pins in a way that does not conflict with p-term allocation.

Table 2. Product Term Allocation Rings

Ring 1			Ring 2		
Current Macrocell	Next Macrocell	Previous Macrocell	Current Macrocell	Next Macrocell	Previous Macrocell
1	2	6	7	8	12
2	3	1	8	9	7
3	4	2	9	10	8
4	5	3	10	11	9
5	6	4	11	12	10
6	1	5	12	7	11

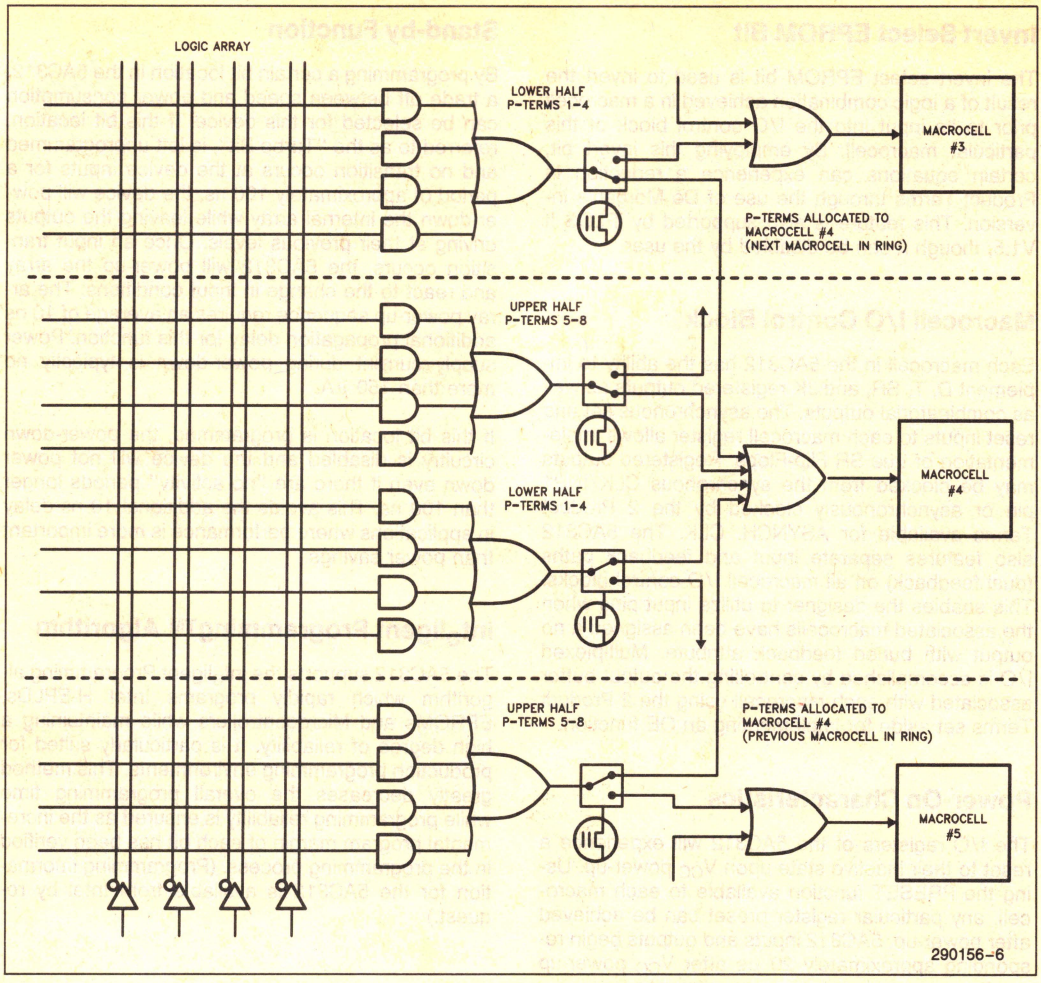


Figure 5. Product Term Allocation (8 + 4 + 4)

Invert Select EPROM Bit

The invert select EPROM bit is used to invert the result of a logic combination achieved in a macrocell prior to its input into the I/O control block of this particular macrocell. By employing this invert bit, certain equations can experience a reduction in Product Terms through the use of De Morgan's inversion. This feature is also supported by iPLDS II V1.5, though it can be disabled by the user.

Macrocell I/O Control Block

Each macrocell in the 5AC312 has the ability to implement D, T, SR, and JK registered outputs as well as combinatorial outputs. The asynchronous set and reset inputs to each macrocell register allows implementation of true SR Flip-Flops. Registered outputs may be clocked from the synchronous CLK/INP1 pin or asynchronously clocked by the 2 Product Terms available for ASYNCH. CLK. The 5AC312 also features separate input and feedback paths (dual feedback) on all macrocell I/O control blocks. This enables the designer to utilize input pins when the associated macrocells have been assigned a no output with buried feedback attribute. Multiplexed I/O is accomplished by controlling the output buffer associated with each macrocell using the 2 Product Terms set aside for implementing an OE function.

Power-On Characteristics

The I/O registers of the 5AC312 will experience a reset to their inactive state upon V_{CC} power-up. Using the PRESET function available to each macrocell, any particular register preset can be achieved after power-up. 5AC312 inputs and outputs begin responding approximately 20 μs after V_{CC} power-up or after a power-loss/power-up sequence.

Stand-by Function

By programming a certain bit location in the 5AC312, a trade off between speed and power consumption can be selected for this device. If this bit location, referred to as the "Turbo Bit", is left unprogrammed and no transition occurs at the device inputs for a period of approximately 100 ns, the device will power-down the internal array while leaving the outputs driving at their previous levels. Once an input transition occurs, the 5AC312 will power-up the array and react to the change in input conditions. The array power-up sequence requires an average of 10 ns additional propagation delay for this function. Power supply current during power-down is typically no more than 150 μA.

If this bit location is programmed, the power-down circuitry is disabled and the device will not power down even if there are "no activity" periods longer than 100 ns. This avoids the additional 10 ns delay in applications where performance is more important than power savings.

int$_e$ligent Programming™ Algorithm

The 5AC312 supports the int$_e$ligent Programming algorithm which rapidly programs Intel H-EPLDs, EPROMs and Microcontrollers while maintaining a high degree of reliability. It is particularly suited for production programming environments. This method greatly decreases the overall programming time while programming reliability is ensured as the incremental program margin of each bit has been verified in the programming process. (Programming information for the 5AC312 is available from Intel by request.)

FUNCTIONAL TESTING

Since the logical operation of the 5AC312 is controlled by EPROM elements, the device is completely testable during the manufacturing process. Each programmable EPROM bit controlling the internal logic is tested using application-independent test patterns. EPROM cells in the 5AC312 are 100% tested for programming and erase. After testing, the devices are erased before shipments to the customers. No post-programming tests of the EPROM array are required.

The testability and reliability of EPROM-based programmable logic devices is an important feature over similar devices based on fuse technology. Fuse-based programmable logic devices require a user to perform post-programming tests to insure device functionality. During the manufacturing process, tests on these parts can only be performed in very restricted manners in order to avoid a pre-programming of the array.

INTEL PROGRAMMABLE LOGIC DEVELOPMENT SYSTEM II (iPLDS II)

Release 1.5 of iPLDS II graphically shown in Figure 6 provides all the tools needed to design with the 5AC312 EPLD. In addition to providing development assistance, iPLDS II insulates the user from having to know all the intricate details of EPLD architecture (the machine will optimize a design to benefit from architectural features). It contains comprehensive third generation software that supports four different design entry methods, minimizes logic, does automatic pin assignments and produces the best design fit for the selected EPLD. It is user friendly with guided menus, on-line Help messages and soft key inputs.

In addition, the iPLDS II contains programmer hardware in the form of an iUP-PC Universal Programmer-Personal Computer to enable the user to program EPLDs, read and verify programmed devices and also to graphically edit programming files. The software generates industry standard JEDEC object code output files which can be downloaded to other programmers as well.

The iPLDS II has interfaces to popular schematic capture packages (including Dash series from FutureNet* and PC-CAPS** from PCAD) to enable designs to be entered using schematics. A more integrated schematic entry method is provided by SCHEMA II-PLD, a low-cost schematic capture package that supports EPLD primitives and user-defined macro symbols. SCHEMA II-PLD contains the EPLD Design Manager, which provides a single user interface to both SCHEMA II-PLD and iPLS II software. The other design formats supported are Boolean equation entry and State Machine design entry.

The iPLDS operates on the IBM† PC/XT, PC/AT, or other compatible machine with the following configuration:

1. At least one floppy disk drive and hard disk drive.
2. MS-DOS†† Operating System Version 3.0 or greater.
3. 640K Memory.
4. Intel iUP-PC Universal Programmer-Personal Computer and GUPI Adaptor (supplied with iPLDS II)
5. A color monitor is suggested.

Detailed information on the Intel Programmable Logic Development System II is contained in a separate Intel data sheet. (Order Number: 280168)

* FutureNet is a registered trademark of FutureNet Corporation. DASH is a trademark of FutureNet Corporation.

** PC-CAPS is a trademark of P-CAD Corporation.

† IBM Personal Computer is a registered trademark of International Business Machines Corporation.

†† MS-DOS is a registered trademark of Microsoft Corporation.

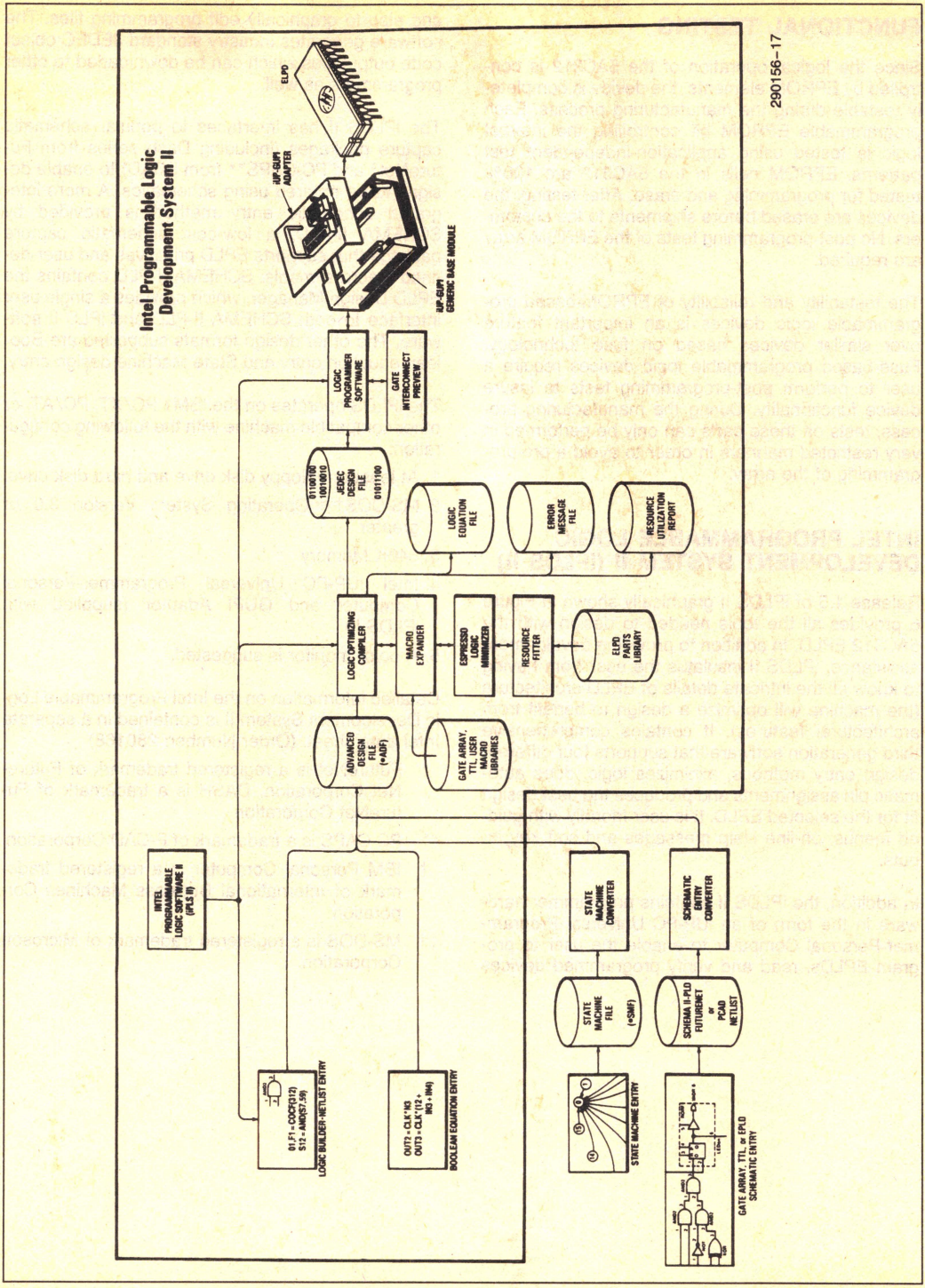


Figure 6. iPLDS II Intel Programmable Logic Development System

ABSOLUTE MAXIMUM RATINGS*

Supply Voltage (V_{CC}) (1) . . . . . −2.0V to +7.0V
Programming Supply Voltage (V_{PP}) (1) . . . . . −2.0V to +13.5V
D.C. Input Voltage (V_I)(1, 2) . . . −0.5V to V_{CC} + 0.5V
Storage Temperature (T_{stg}) . . . . . −65°C to +150°C
Ambient Temperature (T_{amb}) (3) . . −10°C to +85°C

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

NOTES:
1. Voltages with respect to GND.
2. Minimum D.C. input is −0.5V. During transitions, the inputs may undershoot to −2.0V for periods of less than 20 ns under no load conditions.
3. Under bias. Extended temperature range versions are available.

D.C. CHARACTERISTICS T_A = 0°C to +70°C, V_{CC} = 5.0V ±5%

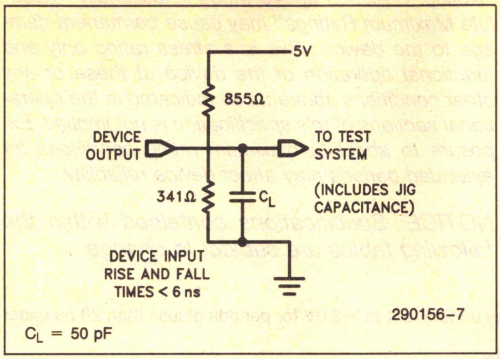
Symbol	Parameter	Min	Typ	Max	Unit	Test Conditions
V_{IH}(4)	High Level Input Voltage	2.0		V_{CC} + 0.3	V	
V_{IL}(4)	Low Level Input Voltage	−0.3		0.8	V	
V_{OH}(5)	High Level Output Voltage	2.4			V	I_O = −4.0 mA D.C., V_{CC} = min.
V_{OL}	Low Level Output Voltage			0.45	V	I_O = 4.0 mA D.C., V_{CC} = min.
I_I	Input Leakage Current			±10	µA	V_{CC} = max., GND < V_{OUT} < V_{CC}
I_{OZ}	Output Leakage Current			±10	µA	V_{CC} = max., GND < V_{OUT} < V_{CC}
I_{SC}(6)	Output Short Circuit Current	−30		−90	mA	V_{CC} = max., V_{OUT} = 0.5V
I_{SB}(7)	Standby Current		150		µA	V_{CC} = max., V_{IN} = V_{CC} or GND, Standby Mode
I_{CC}	Power Supply Current		50		mA	V_{CC} = max., V_{IN} = V_{CC} or GND, No Load, Input Freq. = 1 MHz Active Mode (Turbo = Off), Device Prog. as 12-Bit Ctr.

NOTES:
4. Absolute values with respect to device GND; all over and undershoots due to system or tester noise are included.
5. I_O at CMOS levels (3.84V) = −2 mA.
6. Not more than 1 output should be tested at a time. Duration of that test must not exceed 1 second.
7. With Turbo Bit Off, device automatically enters standby mode approximately 100 ns after last input transition.

CAPACITANCE

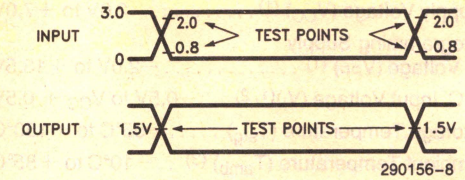
Symbol	Parameter	Min	Typ	Max	Unit	Conditions
C_{IN}	Input Capacitance			20	pF	V_{IN} = 0V, f = 1.0 MHz
C_{OUT}	Output Capacitance			20	pF	V_{OUT} = 0V, f = 1.0 MHz
C_{CLK}	Clock Pin Capacitance			20	pF	V_{OUT} = 0V, f = 1.0 MHz
C_{VPP}	V_{PP} Pin			50	pF	Pin 1

A.C. TESTING LOAD CIRCUIT



C_L = 50 pF

A.C. TESTING INPUT, OUTPUT WAVEFORM



A.C. Testing: Inputs are driven at 3.0V for a Logic "1" and 0V for a Logic "0". Timing Measurements are made at 2.0V for a Logic "1" and 0.8V for a Logic "0" on inputs. Outputs are measured at a 1.5V point.

A.C. CHARACTERISTICS T_A = 0°C to +70°C, V_{CC} = 5.0V ±5%, Turbo Bit "On"[8]

Symbol	From	To	5AC312-25			5AC312-35			Non-[10] Turbo Mode	Unit
			Min	Typ	Max	Min	Typ	Max		
t_{PD1}	Input	Comb. Output		20	25		30	35	+10	ns
t_{PD2}	I/O	Comb. Output		20	25		30	35	+10	ns
t_{PZX}[9]	I or I/O	Output Enable		20	25		30	35	+10	ns
t_{PXZ}[9]	I or I/O	Output Disable		20	25		30	35	+10	ns
t_{CLR}	Asynch. Reset	Q Reset		20	25		30	35	+10	ns
t_{SET}	Asynch. Set	Q Set		20	25		30	35	+10	ns

NOTES:
8. Typical values are at T_A = 25°C, V_{CC} = 5V, Active Mode.
9. t_{PZX} and t_{PXZ} are measured at ±0.5V from steady-state voltage as driven by spec. output load. t_{PXZ} is measured with C_L = 5 pF.
10. If device is operated with Turbo Bit Off (Non-Turbo Mode), increase time by amount shown.

SYNCHRONOUS CLOCK MODE (MACROCELLS) A.C. CHARACTERISTICS

T_A = 0°C to +70°C, V_{CC} = 5.0V ±5%, Turbo Bit On[8]

Symbol	Parameter	5AC312-25			5AC312-35			Non-[10] Turbo Mode	Unit
		Min	Typ	Max	Min	Typ	Max		
f_{MAX}	Max. Frequency $1/t_{SU}$—No Feedback	50	66		40	50		N/A	MHz
f_{CNT}	Max. Count Frequency $1/t_{CNT}$—with Feedback	33	40		25	28.5		N/A	MHz
t_{SU1}	Input Setup Time to CLK	20	15		25	20		+10	ns
t_{SU2}	I/O Setup Time to CLK	20	15		25	20		+10	ns
t_H	I or I/O Hold after CLK High	0			0				ns
t_{CO}	CLK High to Output Valid		10	15		10	15	+10	ns
t_{CNT}	Register Output Feedback to Register Input—Internal Path		25	30		35	40	+10	ns
t_{CH}	CLK High Time	10			12.5			+10	ns
t_{CL}	CLK Low Time	10			12.5			+10	ns

SYNCHRONOUS CLOCK MODE (INPUT STRUCTURE) A.C. CHARACTERISTICS

T_A = 0°C to +70°C, V_{CC} = 5.0V ±5%, Turbo Bit On[8]

Symbol	Parameter	5AC312-25			5AC312-35			Non-[10] Turbo Mode	Unit
		Min	Typ	Max	Min	Typ	Max		
f_{MAXI}	Max. Frequency	40	50		28.5	33		N/A	MHz
t_{SUIR}	Input Register Setup Time before ILE/ICLK ↓	5			5				ns
t_{PLI}[11]	Minimum Input Clock Period		20	25		30	35	+10	ns
t_{COI}	ICLK ↓ to Comb. Output		25	30		35	40	+10	ns
t_{HI}	I Hold after ICLK/ILE ↓	5			5				ns
t_{EOI}	ILE ↑ to Comb. Output		30	35		35	40	+10	ns
t_{CHI}	ILE/ICLK High Time	10			12.5			+10	ns
t_{CLI}	ILE/ICLK Low Time	10			12.5			+10	ns

NOTE:
11. t_{PLI} = Input signal through registers/latch to macrocell register input.

ASYNCHRONOUS CLOCK MODE A.C. CHARACTERISTICS

T_A = 0°C to +70°C, V_{CC} = 5.0V ±5%, Turbo Bit On[8]

Symbol	Parameter	5AC312-25			5AC312-35			Non-[10] Turbo Mode	Unit
		Min	Typ	Max	Min	Typ	Max		
INPUT STRUCTURE									
f_{AMAXI}	Max. Frequency Input Register $1/(t_{ACLI} + t_{ACHI})$	20			16.6			N/A	MHz
t_{ASUI}	Input Register/Latch Setup Time to Asynch. Clock	0			0				ns
t_{AHI}	Input Register/Latch Hold after Asynch. Clock	23	16		30	25		+10	ns
t_{ACOI}	Asynch. ICLK to Output Valid		40	48		50	60	+10	ns
t_{AEOI}	Asynch. ILE ↑ to Comb. Output		45	53		55	65	+10	ns
t_{ACHI}	Asynch. ICLK High Time	25			30			+10	ns
t_{ACLI}	Asynch. ICLK Low Time	25			30			+10	ns
MACROCELLS									
f_{AMAX}	Max. Frequency $1/(t_{ACL} + t_{ACH})$—No Feedback	20			16.6			N/A	MHz
f_{ACNT}	Max. Frequency $1/t_{ACNT}$—with Feedback	18.2			14.3			N/A	MHz
f_{ASU1}	Input Setup Time to Asynch. Clock	7			10			+10	ns

ASYNCHRONOUS CLOCK MODE A.C. CHARACTERISTICS (Continued)

T_A = 0°C to +70°C, V_{CC} = 5.0V ±5%, Turbo Bit On[8]

Symbol	Parameter	5AC312-25			5AC312-35			Non-[10] Turbo Mode	Unit
		Min	Typ	Max	Min	Typ	Max		
MACROCELLS (Continued)									
t_{ASU2}	I/O Setup Time to Asynch. Clock	7			10			+10	ns
t_{AH}	Input or I/O Hold after Asynch. Clock	23	18		30	25		+10	ns
t_{ACO}	Asynch. CLK to Output Valid		30	35		45	50	+10	ns
t_{ACNT}	Register Output Feedback to Register Input—Internal Path		50	55		65	70	+10	ns
t_{ACH}	Asynch. CLK High Time	25			30			+10	ns
t_{ACL}	Asynch. CLK Low Time	25			30			+10	ns

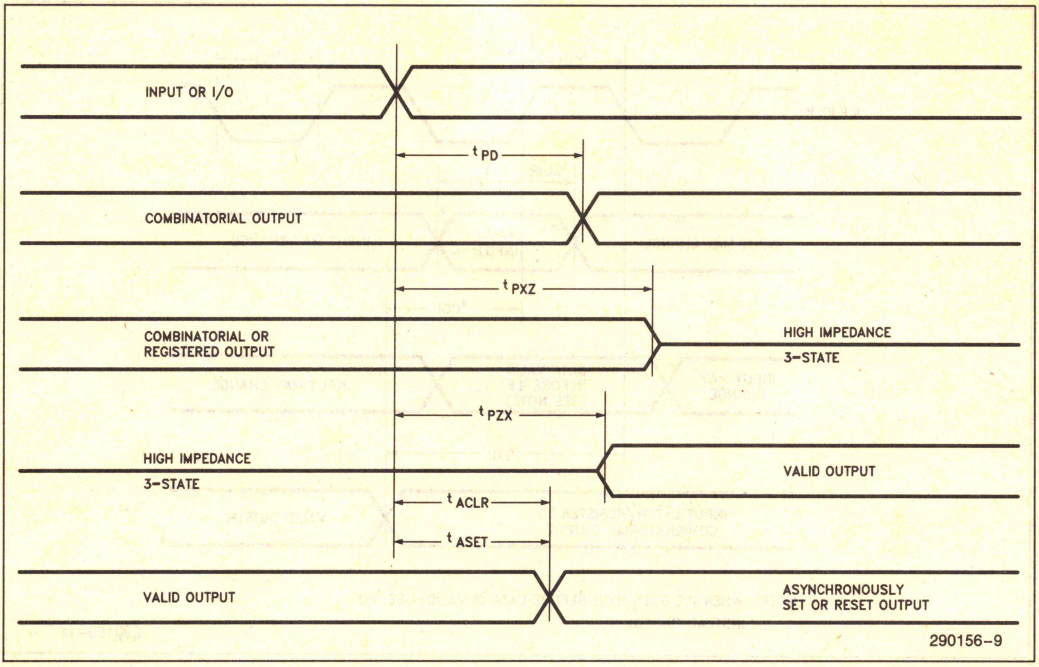
INPUT-CLOCK-TO-MACROCELL-CLOCK A.C. CHARACTERISTICS

T_A = 0°C to +70°C, V_{CC} = 5.0V ±5%, Turbo Bit On[8]

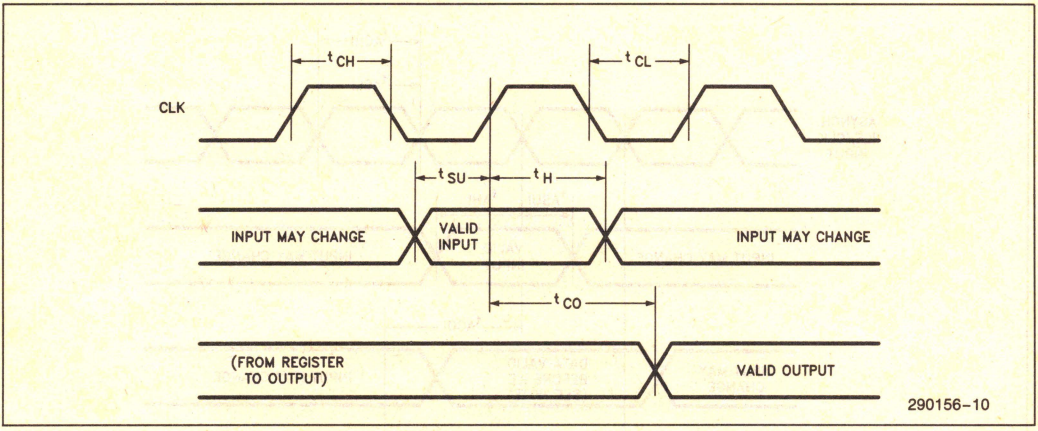
Symbol	Parameter	5AC312-25			5AC312-35			Non-[10] Turbo Mode	Unit
		Min	Typ	Max	Min	Typ	Max		
t_{C1C2}	Synchronous ILE/ICLK Synchronous Macrocell CLK	25			35			+10	ns
	Synchronous ILE/ICLK Asynchronous Macrocell CLK	5			10			+10	ns
	Asynchronous ILE/ICLK Synchronous Macrocell CLK	48			65			+10	ns
	Asynchronous ILE/ICLK Asynchronous Macrocell CLK	20			50			+10	ns

SWITCHING WAVEFORMS

COMBINATORIAL MODE

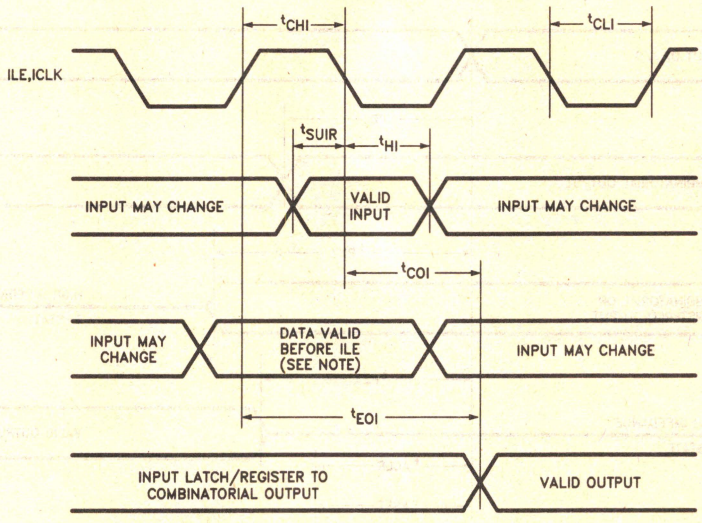


SYNCHRONOUS CLOCK MODE (MACROCELLS)



SWITCHING WAVEFORMS (Continued)

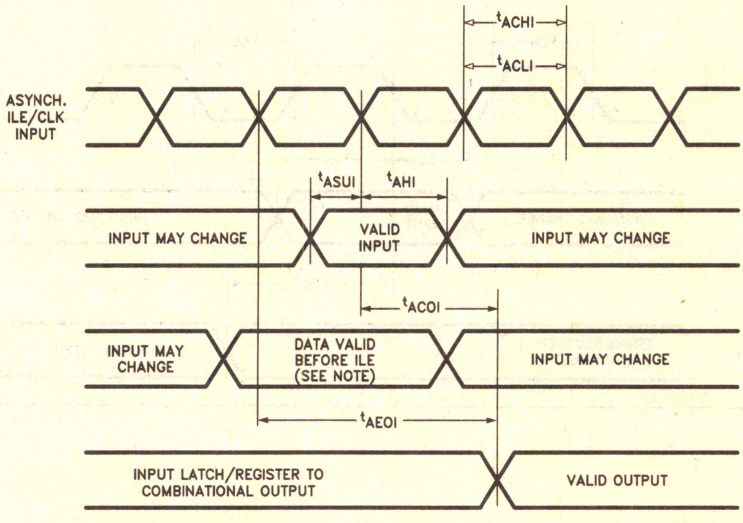
SYNCHRONOUS CLOCK MODE (INPUT STRUCTURE)



NOTE: WHEN ILE GOES HIGH BEFORE DATA IS VALID, USE t_{PD} INSTEAD OF t_{EOI}.

290156-11

ASYNCHRONOUS CLOCK MODE (INPUT STRUCTURE)

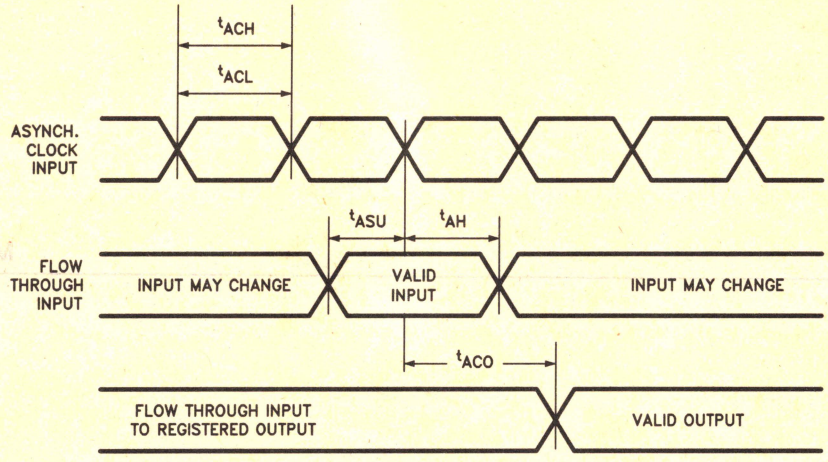


NOTE: WHEN ILE GOES HIGH BEFORE DATA IS VALID, USE t_{PD} INSTEAD OF t_{AEOI}.

290156-13

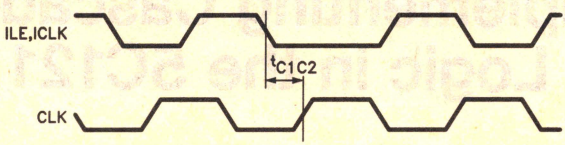
SWITCHING WAVEFORMS (Continued)

ASYNCHRONOUS CLOCK MODE (MACROCELLS)



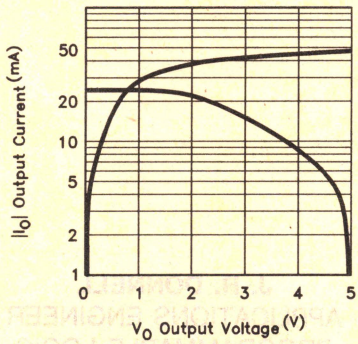
290156-12

INPUT CLOCK-TO-MACROCELL CLOCK TIMING



290156-18

Output Drive Current in Relation to Voltage



Conditions: $T_A = +25°C$

290156-16

intel

APPLICATION BRIEF

AB-8

May 1986

Implementing Cascaded Logic in the 5C121

J. R. DONNELL
APPLICATIONS ENGINEER
PROGRAMMABLE LOGIC

Order Number: 292003-001

PROBLEM

Designs that utilize numerous levels of cascaded logic often result in excessive product terms when expressed in the sum-of-products form. Although this poses no problem when designing with discrete logic, EPLDs are generally optimized for the sum-of-product form. This stems from the architecture of the basic Macrocell.

Macrocells typically consist of a programmable AND array feeding a fixed width OR gate. In the 5C121, OR gate widths range from four to sixteen inputs. For many applications, sixteen available product terms are sufficient. However, one example where product terms become an issue is cascaded exclusive-OR circuits. Here the number of product terms increase as 2**n where n equals the number of exclusive-OR gates. If the number of product terms exceeds sixteen, the equation will not fit directly in the 5C121.

SOLUTION

There is a simple solution to reduce the product term requirements when using cascading XOR (or other) logic. Figure 1 shows a circuit cascading five exclusive ORs. As designed, this circuit expands to 32 product terms when expressed in the minimized sum-of-products form. (This is assuming that signals A thru F are single product terms themselves.) Figure 2 shows the minimized logic equation file produced by Intel's Logic Optimizing Compiler (iLOC).

An easy solution to fitting this logic into the 5C121 is to cascade three exclusive ORs together and then send the result through a No Output Combinational Feedback primitive (NOCF). This signal can now be cascaded through two more XOR's to get the five total. This circuit is shown in Figure 3. Figure 4 shows the logic equation file for this implementation. Note the reduction in product terms from Figure 2. If the buried registers are available, Intel's iPLDs software will automatically assign the combinational feedback to a buried register thereby saving a pin. This technique can be used for any circuit that generates excessive product terms.

The only penalty in this method is the added delay needed for the feedback path. The worst case t_{pd} (input to output delay) for the circuit in Figure 3 would be twice the specified T_{pd} in the 5C121-XX data sheet. Basically the signal must go through the device twice. For the 5C121-90 the T_{pd} would be 180 ns worst case as implemented in Figure 3.

Figure 5 shows the report file generated by the compiler. In this case the NOCF path was automatically assigned to the buried registers.

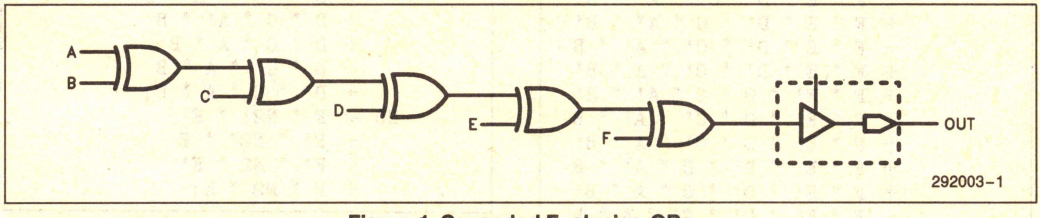


Figure 1. Cascaded Exclusive-ORs

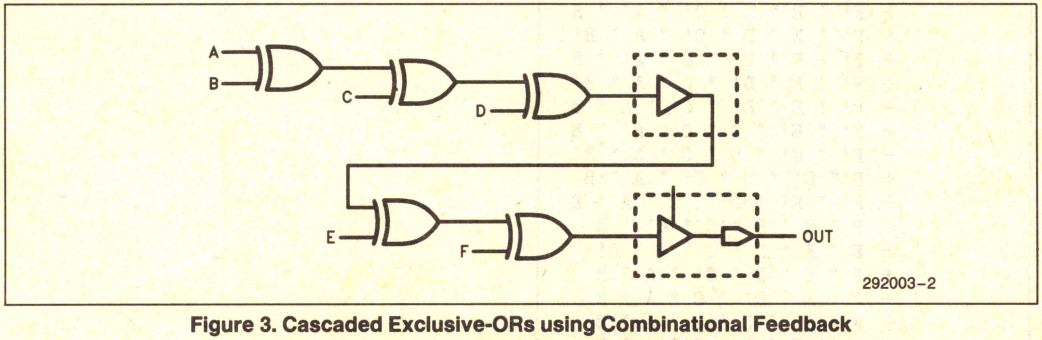


Figure 3. Cascaded Exclusive-ORs using Combinational Feedback

```
5C121
cascading exclusive or's
LB Version 3.0. Baseline 17x, 9/26/85

PART:
      5C121

INPUTS:
      Ap, Bp, Cp, Dp, Ep, Fp

OUTPUTS:
      O

NETWORK:
      A = INP(Ap)
      B = INP(Bp)
      C = INP(Cp)
      D = INP(Dp)
      E = INP(Ep)
      F = INP(Fp)
      O = CONF (NO, VCC)

EQUATIONS:
      NO = F * E' * D' * C' * A' * B'
         + F' * E * D' * C' * A' * B'
         + F' * E' * D * C' * A' * B'
         + F' * E' * D' * C * A' * B'
         + F' * E' * D' * C' * A' * B
         + F' * E' * D' * C' * A * B'
         + F * E * D * C' * A' * B'
         + F * E * D' * C * A' * B'
         + F * E * D' * C' * A' * B
         + F * E * D' * C' * A * B'
         + F * E' * D * C * A' * B'
         + F * E' * D * C' * A' * B
         + F * E' * D * C' * A * B'
         + F * E' * D' * C * A' * B
         + F * E' * D' * C * A * B'
         + F * E' * D' * C' * A * B
         + F' * E * D * C * A' * B'
         + F' * E * D * C' * A' * B
         + F' * E * D * C' * A * B'
         + F' * E * D' * C * A' * B
         + F' * E * D' * C * A * B'
         + F' * E * D' * C' * A * B
         + F' * E' * D * C * A' * B
         + F' * E' * D * C * A * B'
         + F'* E' * D * C' * A * B
         + F' * E' * D' * C * A * B
         + F * E * D * C * A' * B
         + F * E * D * C * A * B'
         + F * E * D * C' * A * B
         + F * E * D' * C * A * B
         + F * E' * D * C * A * B
         + F' * E * D * C * A * B;
```

Figure 2. Minimized Logic Equations for Figure 1

```
5C121
CASCADING 5XORS WITH COMBINATIONAL
FEEDBACK

LB Version 3.0, Baseline 17x, 9/26/85

PART:
      5C121

INPUTS:
      Ap, Bp, Cp, Dp, Ep, Fp

OUTPUTS:
      O

NETWORK:
      A = INP(Ap)
      B = INP(Bp)
      C = INP(Cp)
      D = INP(Dp)
      E = INP(Ep)
      F = INP(Fp)
      O = CONF (NO, VCC)
      N2 = NOCF (N3)

EQUATIONS:
      N3 = D * C' * A' * B'
         + D' * C * A' * B'
         + D' * C' * A' * B
         + D' * C' * A * B'
         + D * C * A' * B
         + D * C * A * B'
         + D * C' * A * B
         + D' * C * A * B;
      NO = F * N2' * E'
         + F' * N2' * E
         + F' * N2 * E'
         + F * N2 * E;
```

Figure 4. Minimized Logic Equations for Figure 3

```
Logic Optimizing Compiler Utilization Report

 ***** Design implemented successfully

JRD
INTEL
October 8, 1985

1
5C121
CASCADING 5XORS WITH COMBINATIONAL FEEDBACK
LB Version 3.0, Baseline 17x, 9/26/85

                5C121
              - - - - -
        GND -| 1   40|- Vcc
        GND -| 2   39|- Vcc
        GND -| 3   38|- Ap
        GND -| 4   37|- Bp
        GND -| 5   36|- Cp
        GND -| 6   35|- Dp
        GND -| 7   34|- Ep
        GND -| 8   33|- Fp
        GND -| 9   32|- O
        GND -|10   31|- RESERVED
        GND -|11   30|- RESERVED
        GND -|12   29|- RESERVED
        GND -|13   28|- GND
        GND -|14   27|- GND
        GND -|15   26|- GND
        GND -|16   25|- GND
        GND -|17   24|- GND
        GND -|18   23|- GND
        GND -|19   22|- GND
        GND -|20   21|- GND
              - - - - -


**INPUTS**

                                                    |          Feeds:
      Name  Pin  Resource   MCell #    PTerms      |  MCells    OE  Clear  Clock

        Fp   33       INP         -         -      |        1     -      -      -

        Ep   34       INP         -         -      |        1     -      -      -

        Dp   35       INP         -         -      |       13     -      -      -

        Cp   36       INP         -         -      |       13     -      -      -

        Bp   37       INP         -         -      |       13     -      -      -

        Ap   38       INP         -         -      |       13     -      -      -


**OUTPUTS**

                                                    |          Feeds:
      Name  Pin  Resource   MCell #    PTerms      |  MCells    OE  Clear

         O   32       CONF        1       4/ 4     |        -     -      -
```

292003-3

```
**BURIED REGISTERS**

                                                    |          Feeds:
      Name   Pin   Resource    MCell #     PTerms    |  MCells     OE   Clear

                -       NOCF         13       8/ 8    |        1      -       -


**UNUSED RESOURCES**

      Name   Pin   Resource      MCell     PTerms

         -     1          -          -          -
         -     2          -          -          -
         -     3          -          -          -
         -     4          -          -          -
         -     5          -          -          -
         -     6          -          -          -
         -     7          -          -          -
         -     8          -         28          4
         -     9          -         27         10
         -    10          -         26          8
         -    11          -         25          6
         -    12          -         24          6
         -    13          -         23          8
         -    14          -         22         10
         -    15          -         21          4
         -    16          -         20         12
         -    17          -         19          4
         -    18          -         18          8
         -    19          -         17          8
         -    21          -         12          8
         -    22          -         11          8
         -    23          -         10          4
         -    24          -          9         12
         -    25          -          8          4
         -    26          -          7         10
         -    27          -          6          8
         -    28          -          5          6
         -    29          -          4          6
         -    30          -          3          8
         -    31          -          2         10
         -    NA          -         14          8
         -    NA          -         15          8
         -    NA          -         16          8

**PART UTILIZATION**

18%      Pins
7%       MacroCells
5%       Pterms
```

292003-4

Figure 5. The Utilization Report

intel

APPLICATION BRIEF

AB-9

May 1986

5C121 As A Three And One Half Digit Display Driver

THOM BOWNS
PROGRAMMABLE LOGIC APPLICATIONS
INTEL CORPORATION

Order Number: 292006-001

INTRODUCTION

Described is a method of constructing a multi-digit, seven segment decoder driver with latching capability in a single EPLD. The design is a simple, easily understood method of using the 5C121 as a seven-segment display driver.

This design has many advantages: (1) the ability to update a single digit without disturbing the others, (2) Outputs are latched and retain their data without update from the controlling device(s), (3) Input interfacing is simple and straightforward, using four data inputs, two digit select lines, and a data strobe line.

The display driver interface is therefore not limited to microprocessor applications only (although it can be used with them). Possible applications include a Multimeter display, a clock or timer display, or a simple controller system display.

PROBLEM

The display driver needs to latch the incoming data at the correct time, route it to the correct digit, and then decode the four bit data into seven-segment output format.

SOLUTION IN EPLD

A simple solution to the display driver imagined above can be realized in the 5C121 EPLD.

The 5C121 EPLD is organized in groups of Macrocells. Each Macrocell contains a number of multiple input AND gates which are feeding an OR gate. The OR gate feeds a selectable registered output. This output may also be routed back into the array for feedback purposes.

Figure 1 shows a basic block diagram of the three and one half digit display driver. The data is input to a distribution block, which sends the data to one of four seven-segment decoders depending upon the digit selected by the Digit Select inputs. The outputs are updated by strobing the WR input. The data input is in a HEX format and may be in the range of 0 to F HEX (0 to 15 Decimal). Digit select is placed upon the two select lines in a binary format; 0, 1, 2, 3. When data is present on the input lines and a digit is selected, the strobe line may be pulsed high and that output digit is then updated to the numeral suggested by the input data.

Figure 2 illustrates the Boolean equivalents of the design in Figure 1. In the NETWORK section of Figure 2, the inputs and outputs of the design are described.

For instance, the NETWORK equation

```
SSA1, SA1F = RORF (ISA1, WRN, GND, GND, VCC)
```

represents that the output pin for segment "A" of the 1st Seven Segment display (SSA1) results from a Registered Output Registered Feedback (RORF) structure in the EPLD. The feedback signal (SA1F) is the same as the signal output (SSA1). The RORF's D input is driven by the signal ISA1, the clock input is driven by WRN, and reset, preset and output enable signals are tied to their default voltage levels (either GND or VCC).

The EQUATION section of Figure 2 shows how the data distribution and decoding logic works. Equations starting with A–G are generic seven segment display equations. Segment decoding results from the combination of the true or false of the four data inputs (e.g., D0 or !D0).

Equations such as

```
SE1 = (E * WE1) + (SE1F * !WE1)
```

show how the data is distributed. Segment E of display 1 (SE1) is valid (ON) if the "E" decode exists and display 1 is chosen by the address inputs (WE1 = !A0 * !A1). It is also valid if it was previously turned on (SE1F) AND seven segment display 1 is not selected (!WE1).

These equations may be entered using LB in the form of a Netlist, or may be entered directly into the ADF by means of a text editor. The ADF is then compiled and programmed into a 5C121 using iPLS.

SUMMARY

This method of using the 5C121 as a three and one half digit display driver is advantageous in respect to its simple interface, and its ability to hold all other digits stable while one is being updated. Displays with more than three and one half digits may be produced in the 5C121 by using the input latches as data storage and by multiplexing the outputs in a scanning fashion.

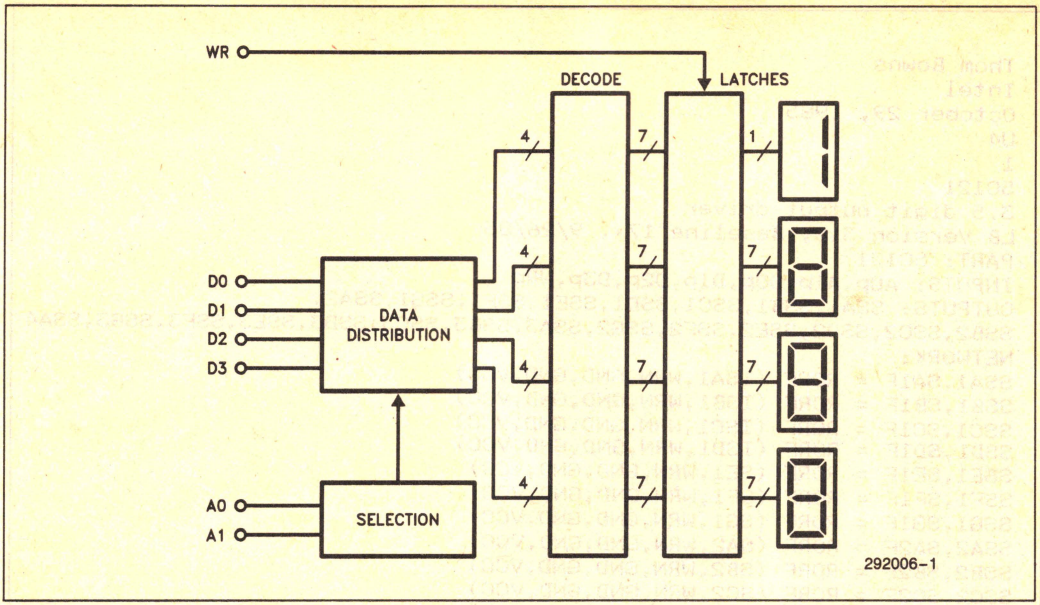


Figure 1. Block Diagram

```
Thom Bowns
Intel
October 29, 1985
U4
1
5C121
3.5 digit output driver
LB Version 3.0, Baseline 17x, 9/26/85
PART: 5C121
INPUTS: AOp,A1p,DOp,D1p,D2p,D3p,WRp
OUTPUTS: SSA1,SSB1,SSC1,SSD1,SSE1,SSF1,SSG1,SSA2,
SSB2,SSC2,SSD2,SSE2,SSF2,SSG2,SSA3,SSB3,SSC3,SSD3,SSE3,SSF3,SSG3,SSA4
NETWORK:
SSA1,SA1F = RORF (ISA1,WRN,GND,GND,VCC)
SSB1,SB1F = RORF (ISB1,WRN,GND,GND,VCC)
SSC1,SC1F = RORF (ISC1,WRN,GND,GND,VCC)
SSD1,SD1F = RORF (ISD1,WRN,GND,GND,VCC)
SSE1,SE1F = RORF (SE1,WRN,GND,GND,VCC)
SSF1,SF1F = RORF (SF1,WRN,GND,GND,VCC)
SSG1,SG1F = RORF (SG1,WRN,GND,GND,VCC)
SSA2,SA2F = RORF (SA2,WRN,GND,GND,VCC)
SSB2,SB2F = RORF (SB2,WRN,GND,GND,VCC)
SSC2,SC2F = RORF (SC2,WRN,GND,GND,VCC)
SSD2,SD2F = RORF (SD2,WRN,GND,GND,VCC)
SSE2,SE2F = RORF (SE2,WRN,GND,GND,VCC)
SSF2,SF2F = RORF (SF2,WRN,GND,GND,VCC)
SSG2,SG2F = RORF (SG2,WRN,GND,GND,VCC)
SSA3,SA3F = RORF (SA3,WRN,GND,GND,VCC)
SSB3,SB3F = RORF (SB3,WRN,GND,GND,VCC)
SSC3,SC3F = RORF (SC3,WRN,GND,GND,VCC)
SSD3,SD3F = RORF (SD3,WRN,GND,GND,VCC)
SSE3,SE3F = RORF (SE3,WRN,GND,GND,VCC)
SSF3,SF3F = RORF (SF3,WRN,GND,GND,VCC)
SSG3,SG3F = RORF (SG3,WRN,GND,GND,VCC)
SSA4,SA4F = RORF (SA4,WRN,GND,GND,VCC)
ISA1 = NOCF (SA1)
ISB1 = NOCF (SB1)
ISC1 = NOCF (SC1)
ISD1 = NOCF (SD1)
WRN = NOT (WR)
WR = INP (WRp)
DO = INP (DOp)
D1 = INP (D1p)
D2 = INP (D2p)
D3 = INP (D3p)
AO = INP (AOp)
A1 = INP (A1p)
EQUATIONS:
A = !D3*!D2*!D1*DO + !D3*D2*!D1*!DO + D3*!D2*D1*DO + D3*D2*!D1*DO;
B = !D3*D2*!D1*DO + D2*D1*!DO + D3*D2*!D1*!DO + D3*D1*DO;
C = !D3*!D2*D1*!DO + D3*D2*!D1*!DO + D3*D2*D1;
D = !D3*!D2*!D1*DO + !D3*D2*!D1*!DO + D2*D1*DO + D3*!D2*D1*!DO;
E = !D3*!D2*DO + !D3*D2*!D1 + !D3*D2*D1*DO + D3*!D2*!D1*DO;
F = !D3*!D2*!D1*DO + !D3*!D2*D1 + !D3*D2*D1*DO + D3*D2*!D1*DO;
G = !D3*!D2*!D1 + !D3*D2*D1*DO + D3*D2*!D1*!DO;
```

292006-2

Figure 2. ADF Listing

```
SE1 = (E * WE1)
        + (SE1F * !WE1);
SF1 = (F * WE1)
        + (SF1F * !WE1);
SG1 = (G * WE1)
        + (SG1F * !WE1);
SA2 = (A * WE2)
        + (SA2F * !WE2);
SB2 = (B * WE2)
        + (SB2F * !WE2);
SC2 = (C * WE2)
        + (SC2F * !WE2);
SD2 = (D * WE2)
        + (SD2F * !WE2);
SE2 = (E * WE2)
        + (SE2F * !WE2);
SF2 = (F * WE2)
        + (SF2F * !WE2);
SG2 = (G * WE2)
        + (SG2F * !WE2);
SA3 = (A * WE3)
        + (SA3F * !WE3);
SB3 = (B * WE3)
        + (SB3F * !WE3);
SC3 = (C * WE3)
        + (SC3F * !WE3);
SD3 = (D * WE3)
        + (SD3F * !WE3);
SE3 = (E * WE3 )
        + (SE3F * !WE3);
SF3 = (F * WE3)
        + (SF3F * !WE3);
SG3 = (G * WE3)
        + (SG3F * !WE3);
SA1 = (A * WE1)
        + (SA1F * !WE1);
SB1 = (B * WE1)
        + (SB1F * !WE1);
SC1 = (C * WE1)
        + (SC1F * !WE1);
SD1 = (D * WE1)
        + (SD1F * !WE1);
SA4 = ((!D3*!D2*!D1*!D0) * WE4) + (SA4F * !WE4);
WE1 = !A0 * !A1;
WE2 = A0 * !A1;
WE3 = !A0 * A1;
WE4 = A0 * A1;
END$
```

292006-3

Figure 2. ADF Listing (Continued)

APPLICATION BRIEF

AB-10

June 1986

Square Pegs in Round Holes—A Fitting Tutorial for the 5C121

J. R. DONNELL
PROGRAMMABLE LOGIC APPLICATIONS
INTEL CORPORATION

Order Number: 292014-001

INTRODUCTION

This application brief explores the various techniques for getting the most out of Intel's line of Erasable Programmable Logic Devices (EPLDs). In many cases, techniques discussed here will not be needed due to the intelligent fitting algorithms built into Intel's Programmable Logic Software (iPLS). As a matter of fact, most designs can be implemented in EPLDs without any knowledge of the device architectures. For complex designs, the designer will still need an in-depth understanding of the target EPLD in order to maximize the EPLD's utility.

This application brief explores fitting techniques for the 5C121, a 1200 gate equivalent CHMOS EPLD. The techniques described here will also apply to any EPLD that supports a similar architecture.

FITTING

When fitting logic designs into the 5C121 there are two typical scenarios: 1) The 5C121 design has been completed without pin assignments and the compiler warns the user that fitting may be time consuming, and 2) pin assignments have been made and the "***ERR-FIT . . . " message comes up.

Let's look at the first situation.

In general, if the designer does not care what signals get assigned to what pins, the choice can be left to the compiler and the compiler will make pin assignments. For simple designs pin assignments are very easy. However, designs that include a variety of different register types, feedback paths, and product term widths may take a long time for the compiler to fit. When the designer is faced with the message, "Fitting may be time consuming", the compilation should be aborted, and intelligent pin assignments made. NOTE: Control C (^C) may be used to abort a design. The software will not stop immediately because the software does not poll the keyboard until it updates the display. Rebooting the system will also work.

To make intelligent pin assignments, the designer needs a basic understanding of the architecture of the part. For the 5C121 this understanding should include the number of product terms supported in each Macrocell, what Macrocells support local feedback, and what Macrocells support global feedback. This information is easily found in the data sheet. One other point, the Macrocells in the 5C121 are grouped into groups of four. All Macrocells in a group must have the same output type. Therefore, if one output is registered, the other three must also be registered. This means that a combinatorial output could not be put into the same group as a registered output. Output enable (OE) terms are also based on Macrocell grouping. All four Macrocells are driven from the same OE term.

Once the basic 5C121 architecture is understood, intelligent pin assignments can be made. After assigning the pins recompile the design using iPLS.

Compiling the design with pin assignments is a new ball game. This time it is fit or not fit. If the design does not fit, an error like: "***ERR-FIT—It is not possible to fit the specific pin requests you made" will occur. In most cases, the compiler will also ask if it can remove pin assignments and try its own. If the design has already been attempted without pin assignments, or if specific pin assignments are needed, answer no and isolate the problem.

ISOLATE THE PROBLEM

The first step towards isolating the problem is to print out a copy of the utilization report (<Filename>.RPT), logic equation file (<Filename>.LEF), and the Advanced Design File (<Filename>.ADF). Next, fill out the 5C121 architecture worksheet included in this application brief. Include the signal name for each pin, the type of output, and the number of product terms needed for each output. All this information is available in the files that were printed earlier. The next step is to identify the conflict.

CONFLICTS

There are three potential conflicts with pin assignments in the 5C121; incompatible output structures, excessive product terms, and local/global feedback conflicts. Incompatible output structures and excessive product term errors are the easiest to spot.

INCOMPATIBLE OUTPUT STRUCTURES

As shown in the 5C121 Design Worksheet, the 5C121 is divided into six Macrocell groupings. The data sheet refers to these as the A-1, B-1, A-2, B-2, A-3, and B-3 Macrocells. One requirement of the 5C121 architecture is that Macrocells within the same grouping have the same output structure. This was discussed earlier, but it is worth revisiting. The file titled example 1 in the appendix shows an ADF for a design that contains such an I/O conflict. Following the ADF is a completed 5C121 architecture worksheet with a number of problems. Concentrating on the incompatible output problem on the 5C121 worksheet, notice that pins 31 and 32 belong to the same Macrocell group, and that they are assigned conflicting I/O structures.

The solution to an incompatible output structure conflict may be as simple as reassigning pins. Another option may be to use a different output type for that sig-

nal. This is very dependent on the design. Another option is possible when a Macrocell grouping has been assigned combinatorial output structure, and a registered output needs to be assigned to that same group. A possible solution is to use one of the buried registers configured as a NORF (No Output Registered Feedback) cell to hold the signal, and then send the signal out through a CONF (Combinatorial Output No Feedback) primitive. This output primitive is compatible with the other output primitives in that grouping, and the register output requirement has also been satisfied. The penalty is loss of speed due to the additional feedback path.

EXCESSIVE PRODUCT TERMS

Excessive product term conflicts are also easy to spot. (A product term consists of a set of signals ANDed together which are separated from other ANDed groups by an OR gate.) Written next to the I/O slot on the 5C121 architecture worksheet is the number of product terms that each Macrocell supports. Match that number with the number of product terms for each output indicated in the logic equation file (LEF). If more product terms are required of a output than are provided, there is a product term conflict. The utilization report also shows the number of product terms used for each signal.

The solution, again, may be as simple as reassigning pins since the 5C121 supports varying product term widths. In fact, the 5C121 supports up to 16 product terms on pins 16 and 24. Note that four of those product terms are shared with the adjacent Macrocell. Sharing means that those signals are common. It is not product term allocation. If the number of product terms exceeds the capability of the device, the design may still fit by splitting up long equations and inserting NOCF (No Output Combinatorial Feedback) primitives. Again the price for using this solution is reduced speed. This technique is covered more thoroughly in AB-8 titled: Implementing Cascaded Logic in the 5C121.

LOCAL/GLOBAL FEEDBACK

It is possible to encounter one other type of fitting conflict in the 5C121. This occurs when a feedback signal from the A-1 or A-2 Macrocells feeds the B-1 or B-2 Macrocells. The issue is that these Macrocells feed busses that are local to one half of the chip. Therefore, the signal is not physically available to the other side of the device.

The best way to understand the local and global bussing in the 5C121 is to divide the chip in half lengthwise. One side contains the A Macrocells, and the other side contains the B Macrocells. The two sides are mirror images. Speaking generically now, the -1 and -2 Macrocells feed only local busses; local to their respective side of the device. The -3 Macrocells and the buried registers feed global busses which route signals to both sides of the device. Therefore a feedback signal coming from the A-1 or A-2 group can only feed the A Macrocells, however, a feedback signal from the A-3 group could feed the B-1, B-2, B-3, or the B buried Macrocells. This local/global bussing applies to both feedback and input signals on the I/O pins. All of the dedicated inputs feed the global bus.

Example 1 also shows a simple two bit counter with seven segment driver outputs. The worksheet shows that the counter registers were assigned to pins 27 and 28, while the seven segment outputs were assigned to pins 8 thru 14. The seven segment outputs decode the feedback signals from the counter registers to generate the appropriate digit output, and therefore must have access to those signals. This presents a local/global feedback conflict. If the designer is locked into those specific pin assignments a design workaround is needed.

One solution might be to take the outputs of the counter and externally tie them to dedicated input pins thereby making those signals global. This would work but that solution ends up wasting input pins. A better solution would be to internally route the counter feedback signals through one of the buried registers configured as a NOCF primitive. After passing through the buried register the signals become global. Both the incompatible output solution and this solution are shown in the worksheet, ADF, and utilization report shown as example 2. If we did not need the counter signals externally, it would of been wise to simply use the buried registers to perform the counting function.

One final comment regarding the utilization report. The utilization report shown in example 1 indicates that signals CLK and CNT feed Macrocell 1001 and 1002. These are fictitious Macrocell numbers that the software assigns to requests that cannot be met. In example 1, three requests were unfulfilled: REGOUT, LED1 and LED0. REGOUT was unfulfilled because of incompatible output structures. LED0 and LED1 were unfulfilled because their feedback signals needed to drive the seven segment display outputs. This was impossible because the LED outputs were assigned to a local bus on the opposite side of the device.

The files shown in example 2 fix the LED fitting problems by sending the feedback signals through the buried registers, thereby making them global. In the case of REGOUT, the buried register primitive NORF (No Output Registered Feedback) is used, allowing the output primitive to be combinatorial.

EXAMPLE 1
ADF

```
JR Donnell
Intel
April 3, 1986

0
5C121
Fitting example
LB Version 3.0, Baseline 17x, 9/26/85
PART: 5C121
INPUTS: CNT@2,CLK@1
OUTPUTS: LED0@28,LED1@27,REGOUT@32,CONFOUT@31,SEGA@8,
         SEGB@9,SEGC@10,SEGD@11,SEGE@12,SEGF@13,SEGG@14
NETWORK:
LED0,A = RORF (NLED0D,CLK,GND,GND,VCC)
LED1,B = RORF (NLED1D,CLK,GND,GND,VCC)
REGOUT = RONF (NREGOUTD,CLK,GND,GND,VCC)
CONFOUT = CONF (NCONFOUTIN,VCC)
SEGA = CONF (NSEGAIN,VCC)
SEGB = CONF (NSEGBIN,VCC)
SEGC = CONF (NSEGCIN,VCC)
SEGD = CONF (NSEGDIN,VCC)
SEGE = CONF (NSEGEIN,VCC)
SEGF = CONF (NSEGFIN,VCC)
SEGG = CONF (NSEGGIN,VCC)
CLK = INP (CLK)
CNT = INP (CNT)
EQUATIONS:
NSEGGIN = 2
        + 3;
2 = B*/A;
3 = A*B;
NLED1D = /A*/B*CNT
       + /A*B*/CNT
       + A*/B*CNT
       + A*B*/CNT;
NLED0D = /A*B*CNT
       + A*/B*/CNT
       + A*/B*CNT
       + A*B*/CNT;
NSEGFIN = 0;
0 = /B*/A;
NSEGEIN = 0
        + 2;
NSEGDIN = 0
        + 2
        + 3;
NSEGCIN = 0
        + 1
        + 3;
1 = /B*A;
NSEGBIN = 0
        + 1
        + 2
        + 3;
NSEGAIN = 0
        + 2
        + 3;
NCONFOUTIN = A*B;
NREGOUTD = /A*/B;
END$
```

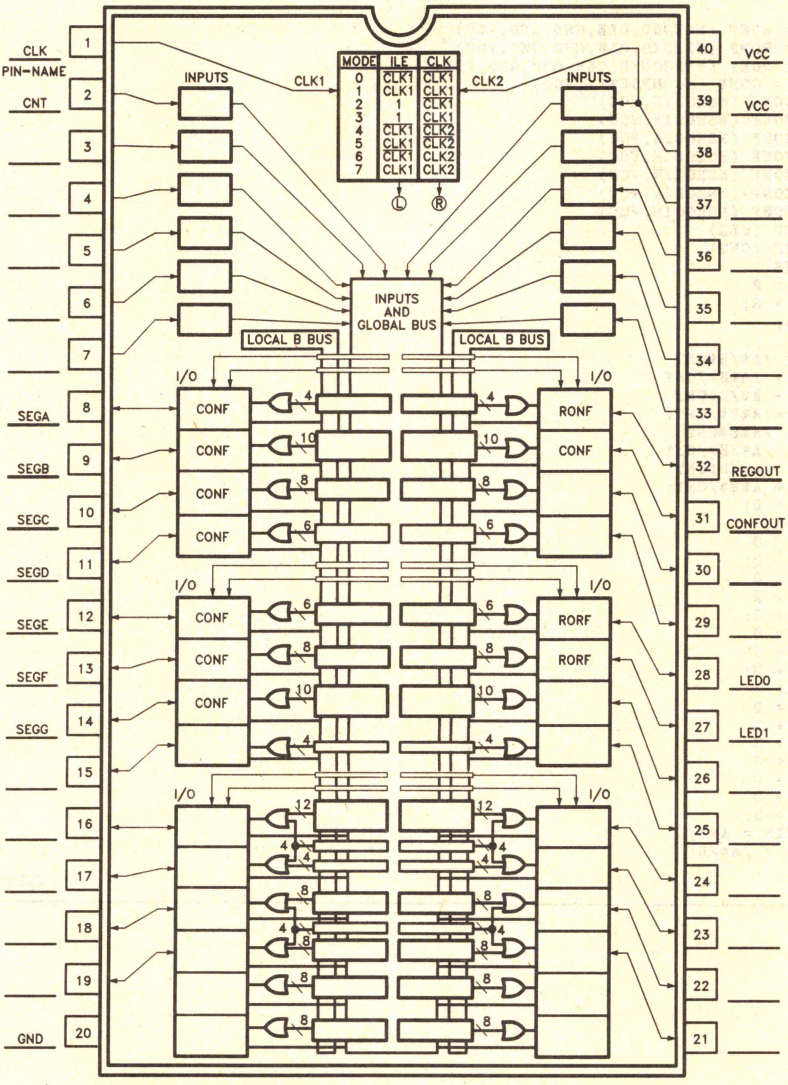
292014-2

SUMMARY

As programmable logic devices become more dense, signal routing and resource partitioning becomes necessary. In general, these choices are made by the semiconductor manufacture to most efficiently utilize the available logic. In some cases though, these choices make certain designs more difficult to implement in a given device. Intelligent software, a basic knowledge of the device architecture, and a little experience in fitting techniques will always make the job easier.

EXAMPLE 1 (Continued)

5C121 Design Worksheet



292014-1

EXAMPLE 1 (Continued)

```
Logic Optimizing Compiler Utilization Report

 ***** Unable to implement design

JR Donnell
Intel
April 3, 1986

0
5C121
Fitting example
LB Version 3.0, Baseline 17x, 9/26/85

                5C121
              - - - - -
        CLK -| 1   40|- Vcc
        CNT -| 2   39|- Vcc
        GND -| 3   38|- GND
        GND -| 4   37|- GND
        GND -| 5   36|- GND
        GND -| 6   35|- GND
        GND -| 7   34|- GND
       SEGA -| 8   33|- GND
       SEGB -| 9   32|- RESERVED
       SEGC -|10   31|- CONFOUT
       SEGD -|11   30|- RESERVED
       SEGE -|12   29|- RESERVED
       SEGF -|13   28|- GND
       SEGG -|14   27|- GND
  RESERVED -|15   26|- GND
        GND -|16   25|- GND
        GND -|17   24|- GND
        GND -|18   23|- GND
        GND -|19   22|- GND
        GND -|20   21|- GND
              - - - - -


**INPUTS**

                                                    |          Feeds:
      Name  Pin  Resource   MCell #    PTerms       |  MCells     OE   Clear  Clock

       CLK    1       INP         -         -       |       -      -      -    Reg

       CNT    2       INP         -         -       |    1001      -      -      -
                                                    |    1002


**OUTPUTS**

                                                    |          Feeds:
      Name  Pin  Resource   MCell #    PTerms       |  MCells     OE   Clear

      SEGA    8      CONF        28       2/ 4      |       -      -      -

      SEGB    9      CONF        27       2/10      |       -      -      -

      SEGC   10      CONF        26       2/ 8      |       -      -      -

      SEGD   11      CONF        25       2/ 6      |       -      -      -
```

292014-3

EXAMPLE 1 (Continued)

```
      SEGE    12       CONF        24       1/ 6          -       -       -
      SEGF    13       CONF        23       1/ 8          -       -       -
      SEGG    14       CONF        22       1/10          -       -       -
   CONFOUT    31       CONF         2       1/10          -       -       -


**UNFULFILLED REQUESTS**

    **OUTPUTS**

                                                                  Feeds:
      Name  Pin  Resource   MCell #   PTerms  :  MCells      OE   Clear
    REGOUT    -      RONF      1000        1  :       -       -       -

      LED1    -      RORF      1001        2  :       2       -       -
                                              :      22
                                              :      23
                                              :      25
                                              :      26
                                              :      28
                                              :    1000
                                              :    1001
                                              :    1002

      LED0    -      RORF      1002        3  :       2       -       -
                                              :      23
                                              :      24
                                              :      25
                                              :      26
                                              :      27
                                              :      28
                                              :    1000
                                              :    1002


**UNUSED RESOURCES**

      Name  Pin  Resource     MCell     PTerms

         -    3         -         -          -
         -    4         -         -          -
         -    5         -         -          -
         -    6         -         -          -
         -    7         -         -          -
         -   15         -        21          4
         -   16         -        20         12
         -   17         -        19          4
         -   18         -        18          8
         -   19         -        17          8
         -   21         -        12          8
         -   22         -        11          8
         -   23         -        10          4
         -   24         -         9         12
         -   25         -         8          4
         -   26         -         7         10
         -   27         -         6          8
         -   28         -         5          6
         -   29         -         4          6
         -   30         -         3          8
         -   32         -         1          4
         -   33         -         -          -
         -   34         -         -          -
         -   35         -         -          -
         -   36         -         -          -
         -   37         -         -          -
         -   38         -         -          -
         -   NA         -        13          8
         -   NA         -        14          8
         -   NA         -        15          8
         -   NA         -        16          8
```

292014-4

292014-5

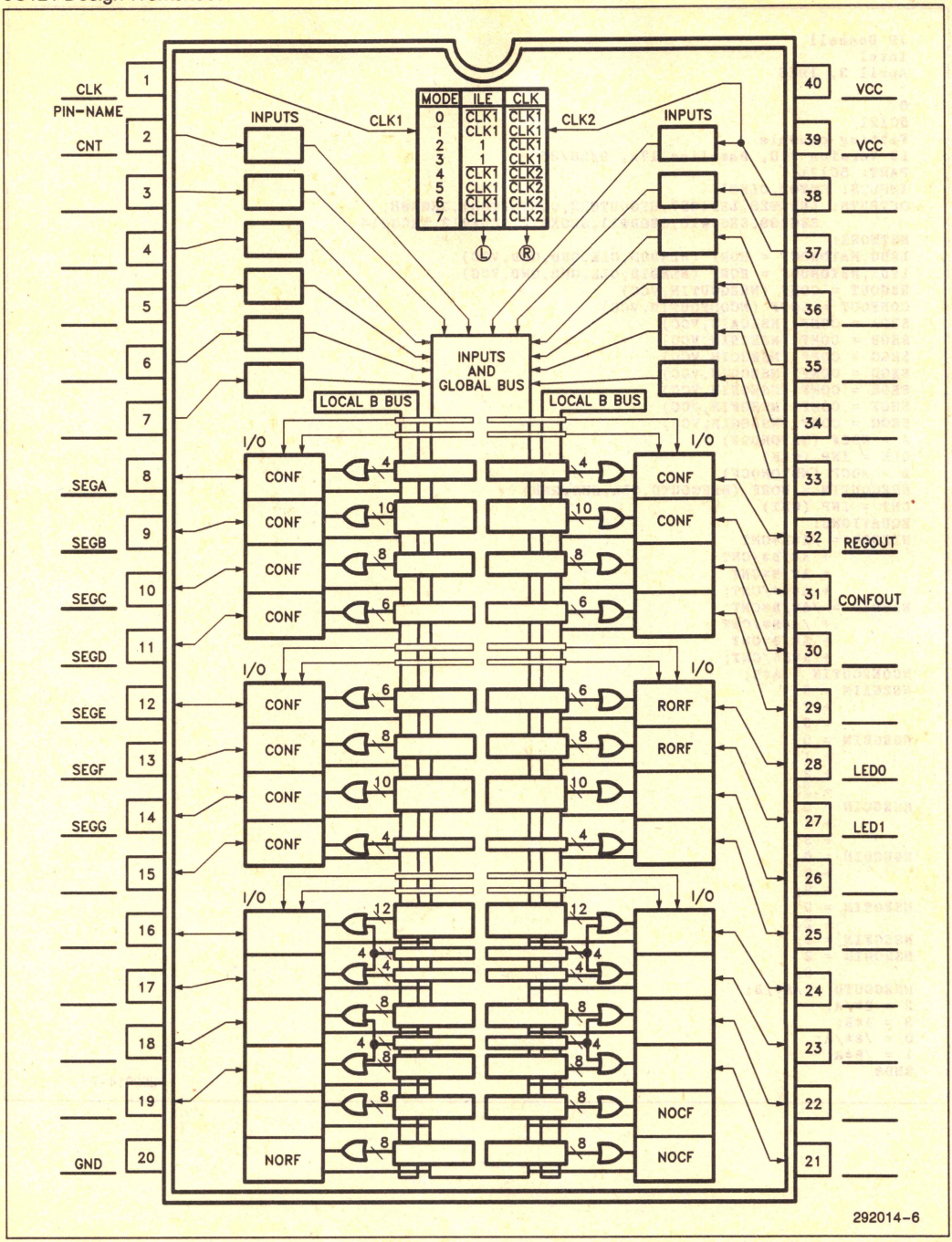
EXAMPLE 2
ADF

```
JR Donnell
Intel
April 3, 1986

0
5C121
Fitting example
LB Version 3.0, Baseline 17x, 9/26/85
PART: 5C121
INPUTS: CNT@2,CLK@1
OUTPUTS: LED0@28,LED1@27,REGOUT@32,CONFOUT@31,SEGA@8,
         SEGB@9,SEGC@10,SEGD@11,SEGE@12,SEGF@13,SEGG@14
NETWORK:
LED0,NATONOCF = RORF (NLED0D,CLK,GND,GND,VCC)
LED1,NBTONOCF = RORF (NLED1D,CLK,GND,GND,VCC)
REGOUT = CONF (NREGOUTIN,VCC)
CONFOUT = CONF (NCONFOUTIN,VCC)
SEGA = CONF (NSEGAIN,VCC)
SEGB = CONF (NSEGBIN,VCC)
SEGC = CONF (NSEGCIN,VCC)
SEGD = CONF (NSEGDIN,VCC)
SEGE = CONF (NSEGEIN,VCC)
SEGF = CONF (NSEGFIN,VCC)
SEGG = CONF (NSEGGIN,VCC)
A = NOCF (NATONOCF)
CLK = INP (CLK)
B = NOCF (NBTONOCF)
NREGOUTIN = NORF (NREGOUTD,CLK,GND,GND)
CNT = INP (CNT)
EQUATIONS:
NLED0D = /A*B*CNT
         + A*/B*/CNT
         + A*/B*CNT
         + A*B*/CNT;
NLED1D = /A*/B*CNT
         + /A*B*/CNT
         + A*/B*CNT
         + A*B*/CNT;
NCONFOUTIN = A*B;
NSEGAIN = 0
         + 2
         + 3;
NSEGBIN = 0
         + 1
         + 2
         + 3;
NSEGCIN = 0
         + 1
         + 3;
NSEGDIN = 0
         + 2
         + 3;
NSEGEIN = 0
         + 2;
NSEGFIN = 0;
NSEGGIN = 2
         + 3;
NREGOUTD = /A*/B;
2 = B*/A;
3 = A*B;
0 = /B*/A;
1 = /B*A;
END$
```

292014-7

EXAMPLE 2 (Continued)

5C121 Design Worksheet



EXAMPLE 2 (Continued)

```
Logic Optimizing Compiler Utilization Report

 ***** Design implemented successfully

JR Donnell
Intel
April 3, 1986

0
5C121
Fitting example
LB Version 3.0, Baseline 17x, 9/26/85

                5C121
              - - - - -
        CLK -¦ 1   40¦- Vcc
        CNT -¦ 2   39¦- Vcc
        GND -¦ 3   38¦- GND
        GND -¦ 4   37¦- GND
        GND -¦ 5   36¦- GND
        GND -¦ 6   35¦- GND
        GND -¦ 7   34¦- GND
       SEGA -¦ 8   33¦- GND
       SEGB -¦ 9   32¦- REGOUT
       SEGC -¦10   31¦- CONFOUT
       SEGD -¦11   30¦- RESERVED
       SEGE -¦12   29¦- RESERVED
       SEGF -¦13   28¦- LED0
       SEGG -¦14   27¦- LED1
   RESERVED -¦15   26¦- RESERVED
        GND -¦16   25¦- RESERVED
        GND -¦17   24¦- GND
        GND -¦18   23¦- GND
        GND -¦19   22¦- GND
        GND -¦20   21¦- GND
              - - - - -


**INPUTS**

                                                    ¦          Feeds:
      Name  Pin  Resource   MCell #    PTerms       ¦  MCells     OE   Clear  Clock

       CLK    1       INP         -         -       ¦      -      -       -    Reg

       CNT    2       INP         -         -       ¦      5      -       -      -
                                                    ¦      6


**OUTPUTS**

                                                    ¦          Feeds:
      Name  Pin  Resource   MCell #    PTerms       ¦  MCells     OE   Clear

      SEGA    8      CONF        28       2/ 4      ¦      -      -       -

      SEGB    9      CONF        27       2/10      ¦      -      -       -

      SEGC   10      CONF        26       2/ 8      ¦      -      -       -

      SEGD   11      CONF        25       2/ 6      ¦      -      -       -
```

292014-8

EXAMPLE 2 (Continued)

```
      SEGE   12      CONF      24      1/ 6        -       -       -
      SEGF   13      CONF      23      1/ 8        -       -       -
      SEGG   14      CONF      22      1/10        -       -       -
      LED1   27      RORF       6      2/ 8       13       -       -
      LED0   28      RORF       5      3/ 6       14       -       -
   CONFOUT   31      CONF       2      1/10        -       -       -
   REGOUT    32      CONF       1      1/ 4        -       -       -


**BURIED REGISTERS**

                                                          Feeds:
      Name  Pin  Resource  MCell #   PTerms    MCells      OE  Clear
                 NOCF        13      1/ 8         2       -       -
                                                  5
                                                  6
                                                 15
                                                 22
                                                 23
                                                 25
                                                 26
                                                 28
                 NOCF        14      1/ 8         2       -       -
                                                  5
                                                 15
                                                 23
                                                 24
                                                 25
                                                 26
                                                 27
                                                 28
                 NORF        15      1/ 8         1       -       -


**UNUSED RESOURCES**

      Name  Pin  Resource   MCell    PTerms

        -    3       -        -         -
        -    4       -        -         -
        -    5       -        -         -
        -    6       -        -         -
        -    7       -        -         -
        -   15       -       21         4
        -   16       -       20        12
        -   17       -       19         4
        -   18       -       18         8
        -   19       -       17         8
        -   21       -       12         8
        -   22       -       11         8
        -   23       -       10         4
        -   24       -        9        12
        -   25       -        8         4
        -   26       -        7        10
        -   29       -        4         6
```

292014-9

EXAMPLE 2 (Continued)

```
        -   30      -      3       8
        -   33      -      -       -
        -   34      -      -       -
        -   35      -      -       -
        -   36      -      -       -
        -   37      -      -       -
        -   38      -      -       -
        -   NA      -     16       8

**PART UTILIZATION**

35%     Pins
50%     MacroCells
10%     Pterms
```

292014-10

APPLICATION BRIEF

AB-11

February 1987

16-Bit Binary Counter Implementation Using the 5C060 EPLD

KARL-HEINZ WEIGL
INTEL CORPORATION
MUNICH, GERMANY

Order Number: 292015-002

INTRODUCTION

System designers often use programmable logic devices to implement counters. Use of PLA devices lets the user build customized counters to suit individual applications. In most cases such counters are not available, 'off-the-shelf' SSI/MSI devices. In other applications, the PLA implementation allows the designer to squeeze the counter function along with other 'glue' tasks into a single PLA, with the attendant higher integration benefits.

Use of traditional 20-pin and 24-pin PLAs, however, does not allow for the construction of large counters having greater than 10 significant bits. This is because these traditional PLAs have register and product term restrictions (even the larger bipolar PLAs have only 8 to 10 registers and less than 8 product terms per register). In contrast, the 5C060 24-pin erasable programmable logic device (EPLD) contains 16 registers that are programmable as 'D', 'T', 'RS' or 'JK' types. These 16 programmable registers enable the construction of Up/Down counters with up to 16 significant bits.

This application brief details the implementation of a 16-bit binary counter in the 5C060 EPLD. The design also demonstrates efficient counter construction utilizing toggle flip-flops (T-FF) that allows for minimum product term utilization.

DESIGN OBJECTIVE

The objective of the design is to implement a counter with the following features: (i) 16-bit binary count, (ii) toggle flip-flops, (iii) asynchronous clear, (iv) RUN/$\overline{\text{STOP}}$ function and (v) UP/$\overline{\text{DOWN}}$ function. The function table is shown in Figure 1.

RESET	UP/DOWN	RUN/STOP	Function
X	X	0	Inhibit Counting
0	0	1	Count Down
0	1	1	Count Up
1	X	X	Reset All Outputs to 'LOW'

Figure 1

TOGGLE FLIP-FLOPS

Counters can be most effectively implemented in PLA architectures using toggle flip-flops. This is because counters constructed with 'D' type flip-flops require an additional product term for every successive significant bit, whereas toggle flip-flop implementation requires only **one** product term per significant bit. Thus, the toggle flip-flop counter design is more miserly in product term consumption than the 'D' register design. Since product term minimization is the key element to maximizing PLA utilization, the T-FF counter design is more efficient. The truth table for the toggle flip-flop is shown in Fig. 2.

T	Q(N)	Q (N + 1)
0	0	0
0	1	1
1	0	1
1	1	0

Figure 2

SOLUTION

The 16-bit binary counter function was implemented in the 5C060 EPLD using the Intel Programmable Logic Development System (iPLDS). The equations for the 16-bit binary counter with the RESET, UP/$\overline{\text{DOWN}}$ and RUN/$\overline{\text{STOP}}$ functions are shown in the 'EQUATIONS' section of the LEF (Fig. 4). The pinout of the 5C060 with the implemented counter is shown in the RPT file (Utilization Report) Fig. 5. This RPT file also shows, under the 'OUTPUTS' section, that in each macrocell only one out of 8 product terms is used. In contrast the same 16-bit counter designed using 'D' type flip-flops would have required more than 16 product terms for the last significant bit.

```
INTEL CORPORATION
JAN. 15, 1987
1
1.0
5C060
BINARY 16-BIT UP/DOWN COUNTER WITH RUN/STOP AND ASYNCH. RESET USING T-FF

LB Version 4.01, Baseline 27.1 4/9/86
OPTIONS: TURBO=ON
PART: 5C060
INPUTS: RS,CLOCK,RESET,UD
OUTPUTS: Q0,Q1,Q2,Q3,Q4,Q5,Q6,Q7,Q8,Q9,QA,QB,QC,QD,QE,QF
NETWORK:
Q0,Q0F = TOTF (Q0T,CLK,CLR,GND,VCC)
Q1,Q1F = TOTF (Q1T,CLK,CLR,GND,VCC)
Q2,Q2F = TOTF (Q2T,CLK,CLR,GND,VCC)
Q3,Q3F = TOTF (Q3T,CLK,CLR,GND,VCC)
Q4,Q4F = TOTF (Q4T,CLK,CLR,GND,VCC)
Q5,Q5F = TOTF (Q5T,CLK,CLR,GND,VCC)
Q6,Q6F = TOTF (Q6T,CLK,CLR,GND,VCC)
Q7,Q7F = TOTF (Q7T,CLK,CLR,GND,VCC)
Q8,Q8F = TOTF (Q8T,CLK,CLR,GND,VCC)
Q9,Q9F = TOTF (Q9T,CLK,CLR,GND,VCC)
QA,QAF = TOTF (QAT,CLK,CLR,GND,VCC)
QB,QBF = TOTF (QBT,CLK,CLR,GND,VCC)
QC,QCF = TOTF (QCT,CLK,CLR,GND,VCC)
QD,QDF = TOTF (QDT,CLK,CLR,GND,VCC)
QE,QEF = TOTF (QET,CLK,CLR,GND,VCC)
QF = TONF (QFT,CLK,CLR,GND,VCC)
Q0T = OR (Q0U,Q0D)
CLK = INP (CLOCK)
CLR = INP (RESET)
Q1T = OR (Q1U,Q1D)
Q2T = OR (Q2U,Q2D)
Q3T = OR (Q3U,Q3D)
Q4T = OR (Q4U,Q4D)
Q5T = OR (Q5U,Q5D)
Q6T = OR (Q6U,Q6D)
Q7T = OR (Q7U,Q7D)
Q8T = OR (Q8U,Q8D)
Q9T = OR (Q9U,Q9D)
QAT = OR (QAU,QAD)
QBT = OR (QBU,QBD)
QCT = OR (QCU,QCD)
QDT = OR (QDU,QDD)
QET = OR (QEU,QED)
QFT = OR (QFU,QFD)
RS = INP (RS)
UD = INP (UD)
NUD = NOT (UD)
Q0U = AND (UD,RS)
```

292015-1

Figure 3. Example .ADF

```
Q1U = AND (UD,Q0F,Q0U)
Q2U = AND (UD,Q1F,Q1U)
Q3U = AND (UD,Q2F,Q2U)
Q4U = AND (UD,Q3F,Q3U)
Q5U = AND (UD,Q4F,Q4U)
Q6U = AND (UD,Q5F,Q5U)
Q7U = AND (UD,Q6F,Q6U)
Q8U = AND (UD,Q7F,Q7U)
Q9U = AND (UD,Q8F,Q8U)
QAU = AND (UD,Q9F,Q9U)
QBU = AND (UD,QAF,QAU)
QCU = AND (UD,QBF,QBU)
QDU = AND (UD,QCF,QCU)
QEU = AND (UD,QDF,QDU)
QFU = AND (UD,QEF,QEU)
NQ0F = NOT (Q0F)
NQ1F = NOT (Q1F)
NQ2F = NOT (Q2F)
NQ3F = NOT (Q3F)
NQ4F = NOT (Q4F)
NQ5F = NOT (Q5F)
NQ6F = NOT (Q6F)
NQ7F = NOT (Q7F)
NQ8F = NOT (Q8F)
NQ9F = NOT (Q9F)
NQAF = NOT (QAF)
NQBF = NOT (QBF)
NQCF = NOT (QCF)
NQDF = NOT (QDF)
NQEF = NOT (QEF)
Q0D = AND (NUD,RS)
Q1D = AND (NUD,NQ0F,Q0D)
Q2D = AND (NUD,NQ1F,Q1D)
Q3D = AND (NUD,NQ2F,Q2D)
Q4D = AND (NUD,NQ3F,Q3D)
Q5D = AND (NUD,NQ4F,Q4D)
Q6D = AND (NUD,NQ5F,Q5D)
Q7D = AND (NUD,NQ6F,Q6D)
Q8D = AND (NUD,NQ7F,Q7D)
Q9D = AND (NUD,NQ8F,Q8D)
QAD = AND (NUD,NQ9F,Q9D)
QBD = AND (NUD,NQAF,QAD)
QCD = AND (NUD,NQBF,QBD)
QDD = AND (NUD,NQCF,QCD)
QED = AND (NUD,NQDF,QDD)
QFD = AND (NUD,NQEF,QED)
END$
```

292015-2

Figure 3. Example .ADF (Continued)

```
INTEL CORPORATION
JAN. 15, 1987
1
1.0
5C060
BINARY 16-BIT UP/DOWN COUNTER WITH RUN/STOP AND ASYNCH. RESET USING T-FF

LB Version 4.01, Baseline 27.1 4/9/86
LEF Version 4.01 Baseline 22.2 2/4/86
OPTIONS: TURBO=ON
PART:
        5C060
INPUTS:
        RS, CLOCK, RESET, UD
OUTPUTS:
        Q0, Q1, Q2, Q3, Q4, Q5, Q6, Q7, Q8, Q9, QA, QB, QC, QD, QE, QF
NETWORK:
        CLK = INP(CLOCK)
        RS = INP(RS)
        CLR = INP(RESET)
        UD = INP(UD)
        Q0, Q0F = TOTF(Q0T, CLK, CLR, GND, VCC)
        Q1, Q1F = TOTF(Q1T, CLK, CLR, GND, VCC)
        Q2, Q2F = TOTF(Q2T, CLK, CLR, GND, VCC)
        Q3, Q3F = TOTF(Q3T, CLK, CLR, GND, VCC)
        Q4, Q4F = TOTF(Q4T, CLK, CLR, GND, VCC)
        Q5, Q5F = TOTF(Q5T, CLK, CLR, GND, VCC)
        Q6, Q6F = TOTF(Q6T, CLK, CLR, GND, VCC)
        Q7, Q7F = TOTF(Q7T, CLK, CLR, GND, VCC)
        Q8, Q8F = TOTF(Q8T, CLK, CLR, GND, VCC)
        Q9, Q9F = TOTF(Q9T, CLK, CLR, GND, VCC)
        QA, QAF = TOTF(QAT, CLK, CLR, GND, VCC)
        QB, QBF = TOTF(QBT, CLK, CLR, GND, VCC)
        QC, QCF = TOTF(QCT, CLK, CLR, GND, VCC)
        QD, QDF = TOTF(QDT, CLK, CLR, GND, VCC)
        QE, QEF = TOTF(QET, CLK, CLR, GND, VCC)
        QF = TONF(QFT, CLK, CLR, GND, VCC)
EQUATIONS:
        QFT = UD' * QEF' * QDF' * QCF' * QBF' * QAF' * Q9F' * Q8F' * Q7F' * Q6F' *
              Q5F' * Q4F' * Q3F' * Q2F' * Q1F' * Q0F' * RS
            + UD * QEF * QDF * QCF * QBF * QAF * Q9F * Q8F * Q7F * Q6F * Q5F *
              Q4F * Q3F * Q2F * Q1F * Q0F * RS;

        QET = UD' * QDF' * QCF' * QBF' * QAF' * Q9F' * Q8F' * Q7F' * Q6F' * Q5F' *
              Q4F' * Q3F' * Q2F' * Q1F' * Q0F' * RS
            + UD * QDF * QCF * QBF * QAF * Q9F * Q8F * Q7F * Q6F * Q5F * Q4F *
              Q3F * Q2F * Q1F * Q0F * RS;

        QDT = UD' * QCF' * QBF' * QAF' * Q9F' * Q8F' * Q7F' * Q6F' * Q5F' * Q4F' *
              Q3F' * Q2F' * Q1F' * Q0F' * RS
            + UD * QCF * QBF * QAF * Q9F * Q8F * Q7F * Q6F * Q5F * Q4F * Q3F *
              Q2F * Q1F * Q0F * RS;
```

292015-3

Figure 4. Example .LEF

```
        QCT = UD' * QBF' * QAF' * Q9F' * Q8F' * Q7F' * Q6F' * Q5F' * Q4F' * Q3F' *
              Q2F' * Q1F' * Q0F' * RS
            + UD * QBF * QAF * Q9F * Q8F * Q7F * Q6F * Q5F * Q4F * Q3F * Q2F *
              Q1F * Q0F * RS;

        QBT = UD' * QAF' * Q9F' * Q8F' * Q7F' * Q6F' * Q5F' * Q4F' * Q3F' * Q2F' *
              Q1F' * Q0F' * RS
            + UD * QAF * Q9F * Q8F * Q7F * Q6F * Q5F * Q4F * Q3F * Q2F * Q1F *
              Q0F * RS;

        QAT = UD' * Q9F' * Q8F' * Q7F' * Q6F' * Q5F' * Q4F' * Q3F' * Q2F' * Q1F' *
              Q0F' * RS
            + UD * Q9F * Q8F * Q7F * Q6F * Q5F * Q4F * Q3F * Q2F * Q1F * Q0F *
              RS;

        Q9T = UD' * Q8F' * Q7F' * Q6F' * Q5F' * Q4F' * Q3F' * Q2F' * Q1F' * Q0F' *
              RS
            + UD * Q8F * Q7F * Q6F * Q5F * Q4F * Q3F * Q2F * Q1F * Q0F * RS;

        Q8T = UD' * Q7F' * Q6F' * Q5F' * Q4F' * Q3F' * Q2F' * Q1F' * Q0F' * RS
            + UD * Q7F * Q6F * Q5F * Q4F * Q3F * Q2F * Q1F * Q0F * RS;

        Q7T = UD' * Q6F' * Q5F' * Q4F' * Q3F' * Q2F' * Q1F' * Q0F' * RS
            + UD * Q6F * Q5F * Q4F * Q3F * Q2F * Q1F * Q0F * RS;

        Q6T = UD' * Q5F' * Q4F' * Q3F' * Q2F' * Q1F' * Q0F' * RS
            + UD * Q5F * Q4F * Q3F * Q2F * Q1F * Q0F * RS;

        Q5T = UD' * Q4F' * Q3F' * Q2F' * Q1F' * Q0F' * RS
            + UD * Q4F * Q3F * Q2F * Q1F * Q0F * RS;

        Q4T = UD' * Q3F' * Q2F' * Q1F' * Q0F' * RS
            + UD * Q3F * Q2F * Q1F * Q0F * RS;

        Q3T = UD' * Q2F' * Q1F' * Q0F' * RS
            + UD * Q2F * Q1F * Q0F * RS;

        Q2T = UD' * Q1F' * Q0F' * RS
            + UD * Q1F * Q0F * RS;

        Q1T = UD' * Q0F' * RS
            + UD * Q0F * RS;

        Q0T = RS;

END$
```

292015-4

Figure 4. Example .LEF (Continued)

```
Logic Optimizing Compiler Utilization Report
FIT Version 4.01 Baseline 27.1 4/9/86

 ***** Design implemented successfully

**** NOTE:  Connect signal CLOCK to  pin 1 AND pin 13.

INTEL CORPORATION
JAN. 15, 1987
1
1.0
5C060
BINARY 16-BIT UP/DOWN COUNTER WITH RUN/STOP AND ASYNCH. RESET USING T-FF

LB Version 4.01, Baseline 27.1 4/9/86
OPTIONS: TURBO=ON

                5C060
              - - - - -
       CLOCK -| 1   24|- Vcc
         GND -| 2   23|- RS
          Q7 -| 3   22|- QF
          Q6 -| 4   21|- QE
          Q5 -| 5   20|- QD
          Q4 -| 6   19|- QC
          Q3 -| 7   18|- QB
          Q2 -| 8   17|- QA
          Q1 -| 9   16|- Q9
          Q0 -|10   15|- Q8
          UD -|11   14|- RESET
         GND -|12   13|- CLOCK
              - - - - -


**INPUTS**

                                                                  Feeds:
      Name  Pin  Resource   MCell #   PTerms    MCells     OE   Clear   Clock

     CLOCK    1       INP         -        -         -      -       -    CLK1
                                                                          CLK2

        UD   11       INP         -        -         1      -       -       -
                                                     2
                                                     3
                                                     4
                                                     5
                                                     6
                                                     7
                                                     8
                                                     9
                                                    10
                                                    11
                                                    12
                                                    13
                                                    14
                                                    15

       GND   12       GND         -        -         -      -       -       -

     CLOCK   13       INP         -        -         -      -       -    CLK1
                                                                          CLK2

     RESET   14       INP         -        -         -      -       1       -
                                                                     2
                                                                     3
                                                                     4
                                                                     5
                                                                     6
                                                                     7
                                                                     8
                                                                     9
                                                                    10
                                                                    11
                                                                    12
                                                                    13
                                                                    14
                                                                    15
                                                                    16
```

292015-5

Figure 5. Example .RPT File

RS	23	INP	-	-	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16	-	-	-
Vcc	24	Vcc	-	-	-	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16	-	-

OUTPUTS

Name	Pin	Resource	MCell #	PTerms	MCells	Feeds: OE	Clear	Clock
Q7	3	TOTF	9	2/ 8	1 2 3 4 5 6 7 8	-	-	-
Q6	4	TOTF	10	2/ 8	1 2 3 4 5 6 7 8 9	-	-	-
Q5	5	TOTF	11	2/ 8	1 2 3 4 5 6 7 8 9 10	-	-	-
Q4	6	TOTF	12	2/ 8	1 2 3 4 5 6 7 8 9 10 11	-	-	-

292015-6

Figure 5. Example .RPT File (Continued)

Q3	7	TOTF	13	2/ 8	1 2 3 4 5 6 7 8 9 10 11 12	-	-	-
Q2	8	TOTF	14	2/ 8	1 2 3 4 5 6 7 8 9 10 11 12 13	-	-	-
Q1	9	TOTF	15	2/ 8	1 2 3 4 5 6 7 8 9 10 11 12 13 14	-	-	-
Q0	10	TOTF	16	1/ 8	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	-	-	-
Q8	15	TOTF	8	2/ 8	1 2 3 4 5 6 7	-	-	-
Q9	16	TOTF	7	2/ 8	1 2 3 4 5 6	-	-	-
QA	17	TOTF	6	2/ 8	1 2 3 4 5	-	-	-

292015-7

Figure 5. Example .RPT File (Continued)

```
        QB   18      TOTF        5      2/ 8        1        -      -      -
                                                     2
                                                     3
                                                     4

        QC   19      TOTF        4      2/ 8        1        -      -      -
                                                     2
                                                     3

        QD   20      TOTF        3      2/ 8        1        -      -      -
                                                     2

        QE   21      TOTF        2      2/ 8        1        -      -      -

        QF   22      TONF        1      2/ 8        -        -      -      -


**UNUSED RESOURCES**

      Name  Pin  Resource     MCell    PTerms
        -    2       -          -         -

**PART UTILIZATION**

95%     Pins
100%    MacroCells
24%     Pterms
```

292015-8

Figure 5. Example .RPT File (Continued)

APPLICATION BRIEF

AB-12

October 1987

Designing a Mailbox Memory for Two 80C31 Microcontrollers Using EPLDs

K. WEIGL & J. STAHL
INTEL CORPORATION
MUNICH, GERMANY

Order Number: 292016-002

INTRODUCTION

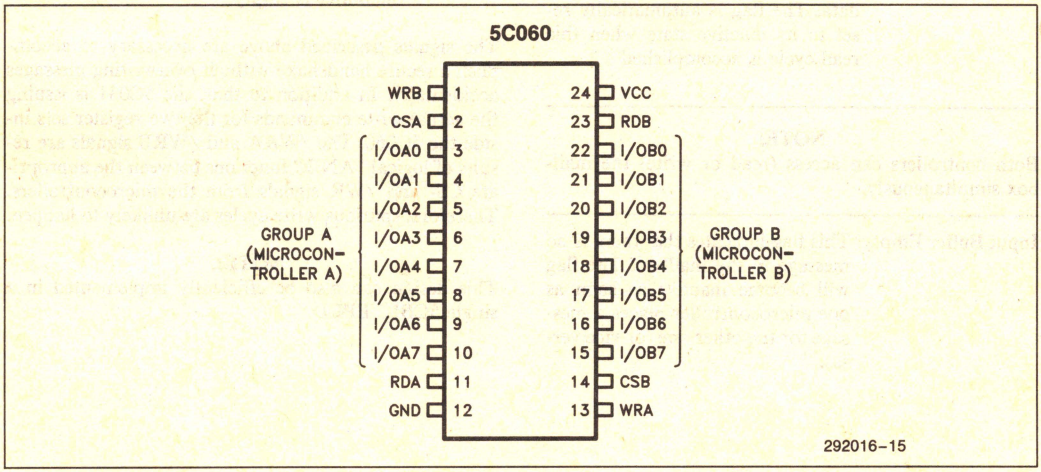
Very often, complex systems involve two or more microcontrollers to fulfill the requirements defined by a given objective. Since the nature of microcontrollers does not allow for easy dual-port memory design (no "READY" input; no "HOLD/HLDA" interface; port-oriented I/O etc.), design engineers are faced with the problem of interchanging information (data and status) between those microcontrollers. This application brief describes the design of a mailbox for exchanging information between two 80C31s, using a 5C060 H-EPLD as a "back-to-back" register, and a 5C031 H-EPLD as an arbitration vehicle to control the actions of the CPUs.

THE 5C060 MAILBOX

In this application, the 16 macrocells of the 5C060 are grouped into two sets of 8 so called "ROIF" (register output with input feedback) primitives to implement the two 8 bit bus interfaces needed. The grouping is done according to the following picture.

The 5C060 allows for independent clocking of 8 macrocells on each side of the chip, the two clock inputs are used to clock data from the microcontroller bus into the chip. To read the data written into the mailbox by one of the controllers, the RDA- (controller A is reading) or RDB- (controller B is reading) line must be pulled low by activating the read command (/RD). In order to avoid spurious read-cycles, the /RD commands from both microcontrollers are logically "ORed" together with an active high CS-signal (Chip Select) inside the 5C060. The CS-signal for both ports is derived from address line A15. Therefore, whenever A15 becomes a logic "1" (true), the mailbox is activated and ready to take or submit data.

Address range for the mailbox: F000 Hex to FFFF Hex
(Upper 12 kbyte)



THE 5C031 "MAILBOX CONTROLLER"

To keep the two microcontrollers informed about the status of their mailbox, the 5C031 is programmed to supply the following signals to both controllers:

```
/OBFA: "OUTPUT BUFFER FULL" FOR MC A

/OBFB: "OUTPUT BUFFER FULL" FOR MC B

/IBEA: "INPUT BUFFER EMPTY" FOR MC A

/IBEB: "INPUT BUFFER EMPTY" FOR MC B

/INTA: INTERRUPT TO MC A

/INTB: INTERRUPT TO MC B
```

The next section will discuss the meanings of these signals in more detail.

Output Buffer Full: This flag is set whenever the controller writes into its own output buffer. The flag remains valid, until the second controller has read the data. The flag is automatically reset to its inactive state when this read cycle is accomplished.

NOTE:
Both controllers can access (read or write) the mailbox simultaneously.

Input Buffer Empty: This flag indicates that there is no message in the mailbox. The flag will become inactive as soon as one microcontroller places a message for the other one (or vice versa).

Example: /IBEA remains "LOW" until microcontroller B places a message for controller A into the mailbox for A. /IBEA will go "HIGH" as soon as controller B has accomplished its write cycle, and will not go "LOW" again until microcontroller A has read the message.

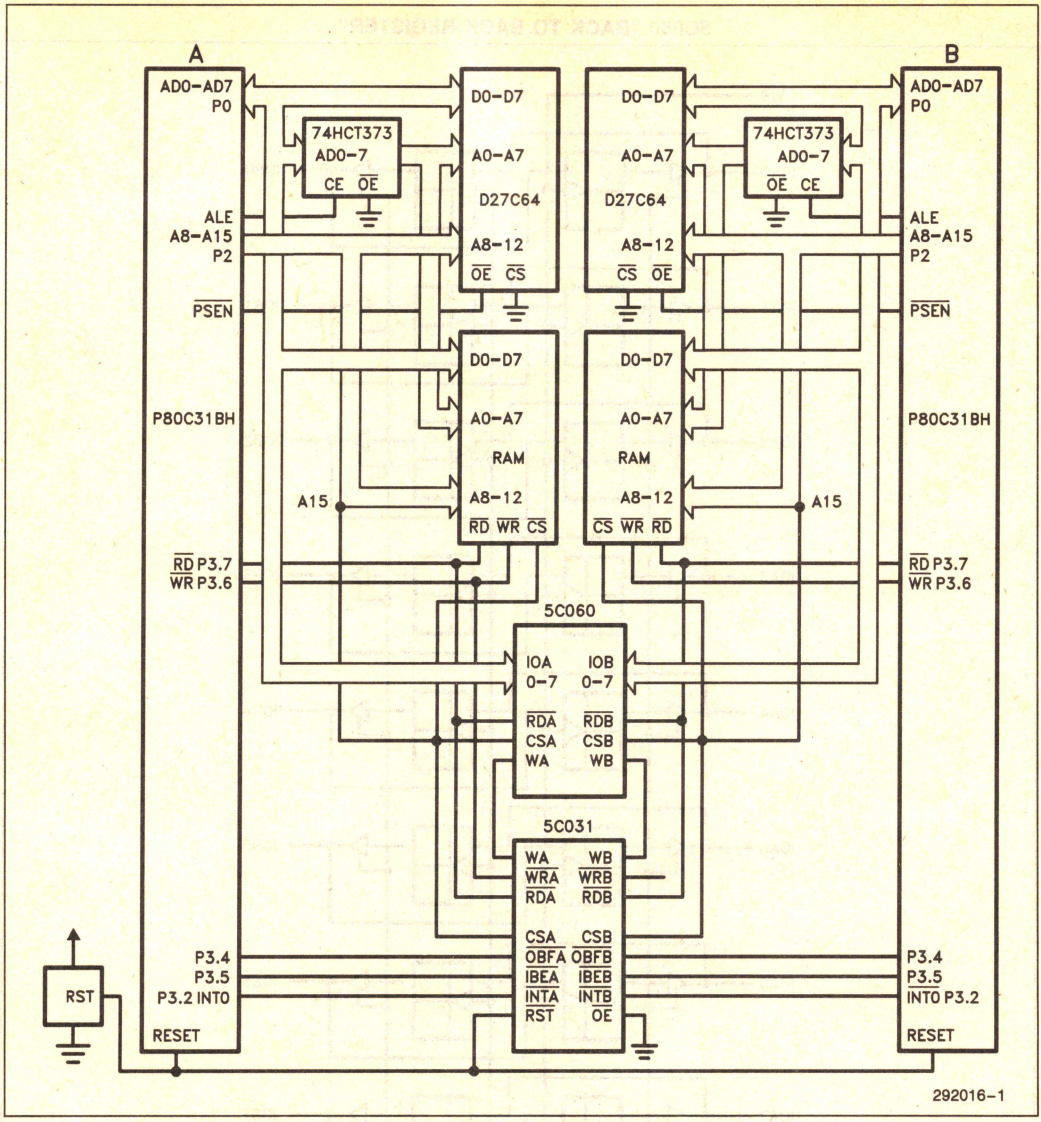
Interrupt: The 5C031 is programmed to supply interrupts to both microcontrollers involved, on one of the following events.

1. The /OBF flag of the opposite microcontroller becomes active; e.g. if controller A is placing a message for controller B, controller B receives an interrupt the same time as /OBFA becomes valid or vice versa.

2. The /IBE flag of the opposite microcontroller goes active, indicating that this controller has received the message; e.g. if controller B reads the message stored by controller A, its /IBEB flag goes active and controller receives an interrupt indicating that the buffer is empty.

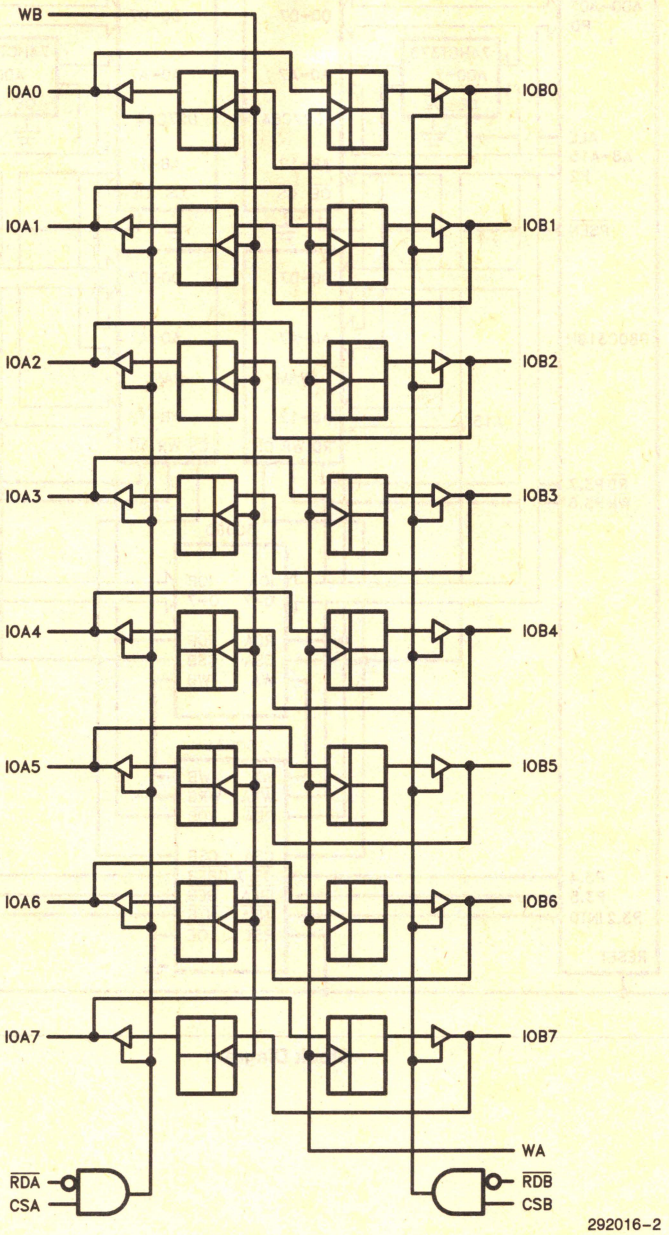
The signals described above are necessary to accomplish a secure handshake without overwriting messages accidentally. In addition to that, the 5C031 is issuing the actual write commands for the two register sets inside the 5C060. The /WRA and /WRB signals are results of logical "AND" functions between the appropriate CS- and /WR signals from the microcontrollers. Therefore, spurious write cycles are unlikely to happen.

NOTE:
This design can also be efficiently implemented in a single 5CBIC EPLD.

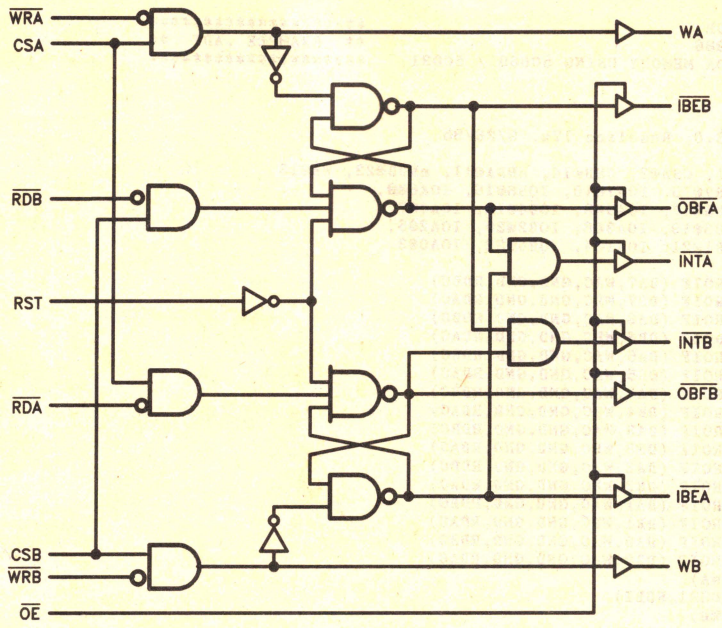


Block Diagram

5C060 "BACK TO BACK REGISTER"



5C031 "MAIL BOX CONTROLLER"



5C060 REGISTER ADF

```
JUERG STAHL
INTEL ZUERICH                                    ********************
March 27, 1986                                   **  EXAMPLE .ADF  **
80C31 MAILBOX MEMORY USING 5C060 / 5C031         ********************
1
5C060

LB Version 3.0, Baseline 17x, 9/26/85
PART: 5C060
INPUTS: WB@1, CSA@2, CSB@14, nRDA@11, nRDB@23, WA@13
OUTPUTS: IOB7@15, IOA7@10, IOB6@16, IOA6@9,
         IOB5@17, IOA5@8, IOB4@18, IOA4@7,
         IOB3@19, IOA3@6, IOB2@20, IOA2@5,
         IOB1@21, IOA1@4, IOB0@22, IOA0@3
NETWORK:
IOB7,DB7 = ROIF (DA7,WAC,GND,GND,RDBC)
IOA7,DA7 = ROIF (DB7,WBC,GND,GND,RDAC)
IOB6,DB6 = ROIF (DA6,WAC,GND,GND,RDBC)
IOA6,DA6 = ROIF (DB6,WBC,GND,GND,RDAC)
IOB5,DB5 = ROIF (DA5,WAC,GND,GND,RDBC)
IOA5,DA5 = ROIF (DB5,WBC,GND,GND,RDAC)
IOB4,DB4 = ROIF (DA4,WAC,GND,GND,RDBC)
IOA4,DA4 = ROIF (DB4,WBC,GND,GND,RDAC)
IOB3,DB3 = ROIF (DA3,WAC,GND,GND,RDBC)
IOA3,DA3 = ROIF (DB3,WBC,GND,GND,RDAC)
IOB2,DB2 = ROIF (DA2,WAC,GND,GND,RDBC)
IOA2,DA2 = ROIF (DB2,WBC,GND,GND,RDAC)
IOB1,DB1 = ROIF (DA1,WAC,GND,GND,RDBC)
IOA1,DA1 = ROIF (DB1,WBC,GND,GND,RDAC)
IOB0,DB0 = ROIF (DA0,WAC,GND,GND,RDBC)
IOA0,DA0 = ROIF (DB0,WBC,GND,GND,RDAC)
WAC = INP (WA)
RDBC = AND(CSBI,RDBI)
WBC = INP (WB)
RDAC = AND(CSAI,RDAI)
CSBI = INP (CSB)
nRDBI = INP(nRDB)
nRDAI = INP(nRDA)
CSAI = INP(CSA)
RDAI = NOT(nRDAI)
RDBI = NOT(nRDBI)

END$
```

292016-4

5C060 REGISTER LEF

```
JUERG STAHL
INTEL ZUERICH                                    **********************
March 27, 1986                                   **  EXAMPLE .LEF  **
80C31 MAILBOX MEMORY USING 5C060 / 5C031         **********************
1
5C060

LB Version 3.0, Baseline 17x, 9/26/85
LEF Version 1.0 Baseline 1.5i 02 Feb 1987
PART:
        5C060
INPUTS:
        WB@1, CSA@2, CSB@14, nRDA@11, nRDB@23, WA@13
OUTPUTS:
        IOB7@15, IOA7@10, IOB6@16, IOA6@9, IOB5@17, IOA5@8, IOB4@18, IOA4@7,
        IOB3@19, IOA3@6, IOB2@20, IOA2@5, IOB1@21, IOA1@4, IOB0@22, IOA0@3
NETWORK:
        WBC = INP(WB)
        WAC = INP(WA)
        CSAI = INP(CSA)
        CSBI = INP(CSB)
        nRDAI = INP(nRDA)
        nRDBI = INP(nRDB)
        IOB7, DB7 = ROIF(DA7, WAC, GND, GND, RDBC)
        IOA7, DA7 = ROIF(DB7, WBC, GND, GND, RDAC)
        IOB6, DB6 = ROIF(DA6, WAC, GND, GND, RDBC)
        IOA6, DA6 = ROIF(DB6, WBC, GND, GND, RDAC)
        IOB5, DB5 = ROIF(DA5, WAC, GND, GND, RDBC)
        IOA5, DA5 = ROIF(DB5, WBC, GND, GND, RDAC)
        IOB4, DB4 = ROIF(DA4, WAC, GND, GND, RDBC)
        IOA4, DA4 = ROIF(DB4, WBC, GND, GND, RDAC)
        IOB3, DB3 = ROIF(DA3, WAC, GND, GND, RDBC)
        IOA3, DA3 = ROIF(DB3, WBC, GND, GND, RDAC)
        IOB2, DB2 = ROIF(DA2, WAC, GND, GND, RDBC)
        IOA2, DA2 = ROIF(DB2, WBC, GND, GND, RDAC)
        IOB1, DB1 = ROIF(DA1, WAC, GND, GND, RDBC)
        IOA1, DA1 = ROIF(DB1, WBC, GND, GND, RDAC)
        IOB0, DB0 = ROIF(DA0, WAC, GND, GND, RDBC)
        IOA0, DA0 = ROIF(DB0, WBC, GND, GND, RDAC)
EQUATIONS:
        RDAC = CSAI * nRDAI';

        RDBC = CSBI * nRDBI';

END$
```

292016-5

5C060 REGISTER UTILIZATION REPORT

```
Logic Optimizing Compiler Utilization Report
FIT Version 1.0 Baseline 1.0i 2/6/87

 ***** Design implemented successfully

JUERG STAHL
INTEL ZUERICH                                  **************************
March 27, 1986                                 **  EXAMPLE .RPT FILE   **
80C31 MAILBOX MEMORY USING 5C060 / 5C031       **************************
1
5C060

LB Version 3.0, Baseline 17x, 9/26/85

              5C060
            - - - - -
        WB -: 1   24 :- Vcc
       CSA -: 2   23 :- nRDB
      IOA0 -: 3   22 :- IOB0
      IOA1 -: 4   21 :- IOB1
      IOA2 -: 5   20 :- IOB2
      IOA3 -: 6   19 :- IOB3
      IOA4 -: 7   18 :- IOB4
      IOA5 -: 8   17 :- IOB5
      IOA6 -: 9   16 :- IOB6
      IOA7 -:10   15 :- IOB7
      nRDA -:11   14 :- CSB
       GND -:12   13 :- WA
            - - - - -


**INPUTS**

                                                                  :          Feeds:
      Name  Pin  Resource   MCell #    PTerms                    :  MCells    OE   Clear  Clock

        WB    1       INP         -         -                    :      -     -       -   CLK1

       CSA    2       INP         -         -                    :      -     9       -      -
                                                                  :            10
                                                                  :            11
                                                                  :            12
                                                                  :            13
                                                                  :            14
                                                                  :            15
                                                                  :            16

      nRDA   11       INP         -         -                    :      -     9       -      -
                                                                  :            10
                                                                  :            11
                                                                  :            12
                                                                  :            13
                                                                  :            14
                                                                  :            15
                                                                  :            16

       GND   12       GND         -         -                    :      -     -       1      -
                                                                  :                    2
                                                                  :                    3
                                                                  :                    4
                                                                  :                    5
                                                                  :                    6
                                                                  :                    7
                                                                  :                    8
                                                                  :                    9
```

292016-6

5C060 REGISTER UTILIZATION REPORT (Continued)

							10 11 12 13 14 15 16		
WA	13	INP	-	-	-	-	-	CLK2	
CSB	14	INP	-	-	-	1 2 3 4 5 6 7 8	-	-	
nRDB	23	INP	-	-	-	1 2 3 4 5 6 7 8	-	-	
Vcc	24	Vcc	-	-	-	-	-	-	

OUTPUTS

Name	Pin	Resource	MCell #	PTerms	MCells	Feeds: OE	Clear	Clock	
IOA0	3	ROIF	9	1/ 8	1	-	-	-	
IOA1	4	ROIF	10	1/ 8	2	-	-	-	
IOA2	5	ROIF	11	1/ 8	3	-	-	-	
IOA3	6	ROIF	12	1/ 8	4	-	-	-	
IOA4	7	ROIF	13	1/ 8	5	-	-	-	
IOA5	8	ROIF	14	1/ 8	6	-	-	-	
IOA6	9	ROIF	15	1/ 8	7	-	-	-	
IOA7	10	ROIF	16	1/ 8	8	-	-	-	
IOB7	15	ROIF	8	1/ 8	16	-	-	-	
IOB6	16	ROIF	7	1/ 8	15	-	-	-	
IOB5	17	ROIF	6	1/ 8	14	-	-	-	
IOB4	18	ROIF	5	1/ 8	13	-	-	-	
IOB3	19	ROIF	4	1/ 8	12	-	-	-	
IOB2	20	ROIF	3	1/ 8	11	-	-	-	
IOB1	21	ROIF	2	1/ 8	10	-	-	-	292016-7
IOB0	22	ROIF	1	1/ 8	9	-	-	-	

All Resources used

PART UTILIZATION

100% Pins
100% MacroCells
12% Pterms

292016-8

5C031 ARBITER ADF

```
JUERG STAHL
INTEL ZUERICH                                    ********************
March 28, 1986                                   **  EXAMPLE .ADF  **
80C31 MAILBOX MEMORY USING 5C060 / 5C031         ********************
2
5C031

LB Version 3.0, Baseline 17x, 9/26/85
PART: 5C031
INPUTS: RST,nWRA,nRDB,CSA,nRDA,nWRB,CSB,nOE
OUTPUTS: WA,nOBFA,nIBEB,nINTA,nINTB,nOBFB,nIBEA,WB
NETWORK:
nWRA = INP(nWRA)
nRDB = INP(nRDB)
RST = INP(RST)
CSA = INP(CSA)
nRDA = INP(nRDA)
nWRB = INP(nWRB)
CSB = INP(CSB)
nOE = INP(nOE)
WRA = NOT(nWRA)
WRB = NOT(nWRB)
RDA = NOT(nRDA)
RDB = NOT(nRDB)
OE = NOT(nOE)
nRST = NOT(RST)
WA = CONF(WAd,VCC)
WAd = AND(CSA,WRA)
WB = CONF(WBd,VCC)
WBd = AND(CSB,WRB)
nRB = NAND(RDB,CSB)
nRA = NAND(RDA,CSA)
nWAd = NOT(WAd)
nWBd = NOT(WBd)
nOBFA,nOBFA = COCF(nOBFAd,OE)
nOBFB,nOBFB = COCF(nOBFBd,OE)
nIBEA,nIBEA = COCF(nIBEAd,OE)
nIBEB,nIBEB = COCF(nIBEBd,OE)
nINTA = CONF(nINTAd,OE)
nINTB = CONF(nINTBd,OE)
nINTAd = AND(nOBFA,nIBEA)
nINTBd = AND(nOBFB,nIBEB)
nOBFBd = NAND(nRA,nIBEA,nRST)
nOBFAd = NAND(nRB,nIBEB,nRST)
nIBEBd = NAND(nWAd,nOBFA)
nIBEAd = NAND(nWBd,nOBFB)

END$
```

292016-9

5C031 ARBITER LEF

```
JUERG STAHL
INTEL ZUERICH                                    ********************
March 28, 1986                                   **  EXAMPLE .LEF  **
80C31 MAILBOX MEMORY USING 5C060 / 5C031         ********************
2
5C031

LB Version 3.0, Baseline 17x, 9/26/85
LEF Version 1.0 Baseline 1.5i 02 Feb 1987
PART:
        5C031
INPUTS:
        RST, nWRA, nRDB, CSA, nRDA, nWRB, CSB, nOE
OUTPUTS:
        WA, nOBFA, nIBEB, nINTA, nINTB, nOBFB, nIBEA, WB
NETWORK:
        RST = INP(RST)
        nWRA = INP(nWRA)
        nRDB = INP(nRDB)
        CSA = INP(CSA)
        nRDA = INP(nRDA)
        nWRB = INP(nWRB)
        CSB = INP(CSB)
        nOE = INP(nOE)
        WA = CONF(WAd, VCC)
        nOBFA, nOBFA = COCF(nOBFAd, OE)
        nIBEB, nIBEB = COCF(nIBEBd, OE)
        nINTA = CONF(nINTAd, OE)
        nINTB = CONF(nINTBd, OE)
        nOBFB, nOBFB = COCF(nOBFBd, OE)
        nIBEA, nIBEA = COCF(nIBEAd, OE)
        WB = CONF(WBd, VCC)
EQUATIONS:
        WBd = CSB * nWRB';

        nIBEAd = CSB * nWRB'
               + nOBFB';

        nOBFBd = (nIBEA * RST' * CSA'
               +  nIBEA * RST' * nRDA)';

        nINTBd = nOBFB * nIBEB;

        nINTAd = nOBFA * nIBEA;

        nIBEBd = CSA * nWRA'
               + nOBFA';

        OE = nOE';

        nOBFAd = (nIBEB * RST' * CSB'
               +  nIBEB * RST' * nRDB)';

        WAd = CSA * nWRA';

END$
```

292016-10

5C031 ARBITER LEF (Continued)

```
Logic Optimizing Compiler Utilization Report
FIT Version 1.0 Baseline 1.0i 2/6/87

 ***** Design implemented successfully

JUERG STAHL                                        **************************
INTEL ZUERICH                                      **  EXAMPLE .RPT FILE   **
March 28, 1986                                     **************************
80C31 MAILBOX MEMORY USING 5C060 / 5C031
2
5C031

LB Version 3.0, Baseline 17x, 9/26/85

               5C031
             _ _ _ _ _
       GND -: 1   20 :- Vcc
       GND -: 2   19 :- WB
       nOE -: 3   18 :- WA
       CSB -: 4   17 :- nOBFB
      nWRB -: 5   16 :- nINTB
      nRDA -: 6   15 :- nINTA
       CSA -: 7   14 :- nIBEB
      nRDB -: 8   13 :- nOBFA
      nWRA -: 9   12 :- nIBEA
       GND -:10   11 :- RST
             - - - - -


**INPUTS**

                                                                  :          Feeds:
      Name   Pin   Resource   MCell #    PTerms   :   MCells      OE   Clear Preset

       nOE     3        INP          -         -   :         -       3     -      -
                                                   :                 4
                                                   :                 5
                                                   :                 6
                                                   :                 7
                                                   :                 8

       CSB     4        INP          -         -   :         1       -     -      -
                                                   :                 7
                                                   :                 8

       nWRB    5        INP          -         -   :         1       -     -      -
                                                   :                 8

       nRDA    6        INP          -         -   :         3       -     -      -

       CSA     7        INP          -         -   :         2       -     -      -
                                                   :                 3
                                                   :                 6

       nRDB    8        INP          -         -   :         7       -     -      -

       nWRA    9        INP          -         -   :         2       -     -      -
                                                   :                 6

       GND    10        GND          -         -   :         -       -     -      -

       RST    11        INP          -         -   :         3       -     -      -
                                                   :                 7

       Vcc    20        Vcc          -         -   :         -       1     -      -
                                                   :                 2
```

292016-11

5C031 ARBITER UTILIZATION REPORT

OUTPUTS

Name	Pin	Resource	MCell #	PTerms	MCells	Feeds: OE	Clear	Preset
nIBEA	12	COCF	8	2/ 8	3 5	-	-	-
nOBFA	13	COCF	7	2/ 8	5 6	-	-	-
nIBEB	14	COCF	6	2/ 8	4 7	-	-	-
nINTA	15	CONF	5	1/ 8	-	-	-	-
nINTB	16	CONF	4	1/ 8	-	-	-	-
nOBFB	17	COCF	3	2/ 8	4 8	-	-	-
WA	18	CONF	2	1/ 8	-	-	-	-
WB	19	CONF	1	1/ 8	-	-	-	-

UNUSED RESOURCES

Name	Pin	Resource	MCell	PTerms
-	1	-	-	-
-	2	-	-	-

PART UTILIZATION

88% Pins
100% MacroCells
18% Pterms

292016-12

APPLICATION BRIEF

AB-16

October 1987

Atypical Latch/Register Construction in EPLDs

THOM BOWNS
PROGRAMMABLE LOGIC APPLICATIONS
INTEL CORPORATION

Order Number: 292031-002

ATYPICAL LATCH/REGISTER CONSTRUCTION IN EPLDs

Though Intel's EPLDs include many of the typical latch and register types, some logic designs require register of latch configurations not directly supported in the current EPLDs. In many cases these register and latch configurations can be generated using the logic array and combinational feedback. A "latch" is defined as a level-triggered, flow-through type such as the 74373, and a "register" is defined as an edge-triggered flip-flop such as the 7474.

This application brief will detail the construction of a D-type latch, an RS latch and a D flip-flop using combinational logic and feedback. Also discussed is the construction of an RS flip-flop, a JK flip-flop and a T flip-flop using registered logic and feedback.

The RS latch is the simplest latch configuration. The equations for it are as follows: QB = !(Q + S), Q = !(QB + R) where Q is the output of one NOR gate, and QB is the output of the other (Note: as a convention in this Ap brief, the "!" operator is used to signify inversion). The schematic of the RS latch is shown in Figure 1a.

Since cross coupled logic is not supported in EPLDs, we must convert the equation to a single term with feedback.

```
Q0, QF = COCF (Q, VCC)
Q = S + !R * QF;
```

where QF is the feedback from Q output.

This circuit can be implemented in an EPLD macrocell. Where combinational feedback is not supported, I/O feedback will suffice. The schematic of this implementation is shown in Figure 1b.

With the RS latch, the inputs are normally low. A logical one on S sets Q to 1, and a one on R resets Q to a 0. Logical ones on both inputs simultaneously cause the output to remain at a high level since S takes precedence over R in this implementation.

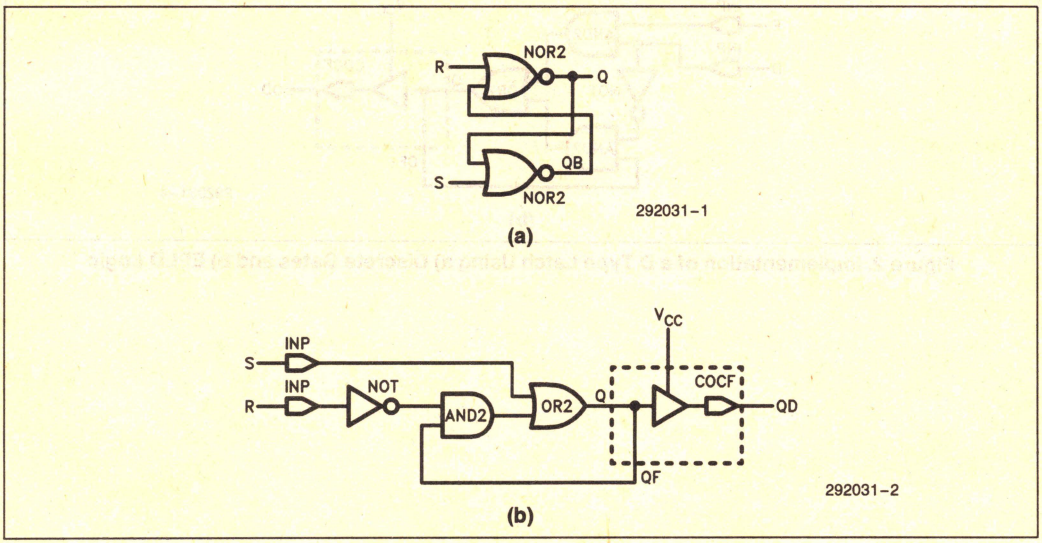


Figure 1. RS Latch Implementation In a) Discrete Gates and b) EPLD Logic

Another latch is the 74373 type, or D latch. This latch works by either enabling input data to appear at the output, or by holding the output to the last input data state. Its equation is this: QB = !(!(!D*E)*Q), Q = !(!(O*E)*QB). Again, Q is the output of one NAND gate, and QB is the output of the other. Figure 2a shows this version of the design.

Again, we must convert to an EPLD-type equation and schematic:

```
Q0, QF = COCF (Q,VCC)
Q = D * E + !E * QF;
```

QF is the feedback from the COCF. In this circuit, when E is high, data flows through transparently. When E is brought low, data is latched. When using input feedback, care must be taken when tri-stating the output as data will no longer be latched. The EPLD implementation is given in Figure 2b.

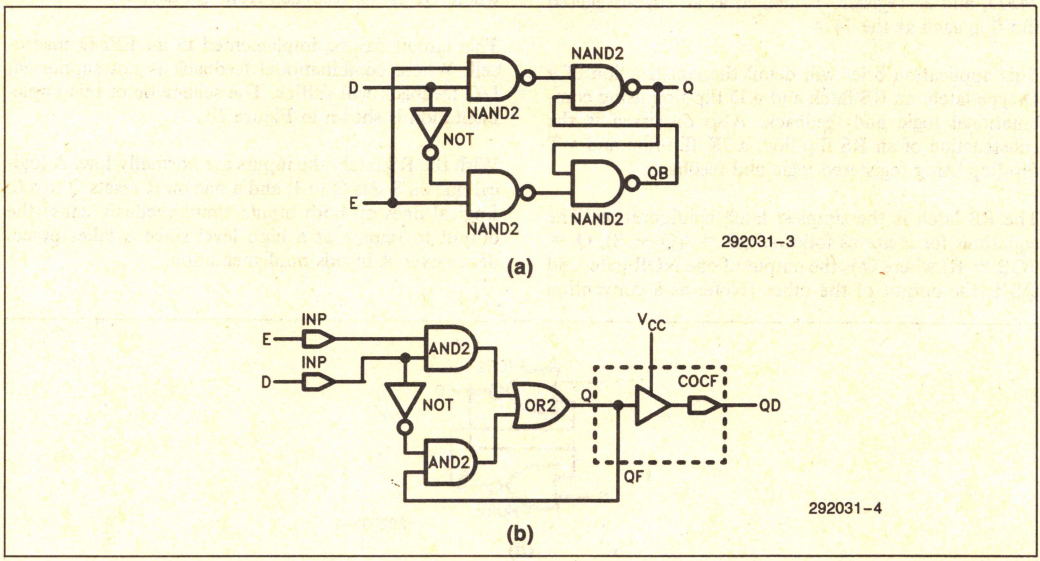


Figure 2. Implementation of a D Type Latch Using a) Discrete Gates and b) EPLD Logic

This latch can be cascaded with a second latch to produce an edge triggered, master/slave D flip-flop, using combinational logic. The flip-flop is a solution to using asynchronous clocking, preset and clear functions when they aren't supported. Also, if an I/O conflict exists within a macrocell group when using registered logic, this design will fit since it uses combinational logic. Figure 3 shows the schematic for this design.

This design does consume two macrocells, but in many cases, that isn't a problem.

The boolean equation of the D flip-flop is this:

```
Q0,QF = COCF (Q,VCC)
YF = NOCF (Y)
Y = D * !CLOCK + YF * CLOCK;
Q = YF * CLOCK + QF * !CLOCK;
```

Q is the flip-flop output and Y is the first latch output. Data is latched in to the second latch on the low-going edge of clock, and is clocked out to Q on the high-going edge of clock.

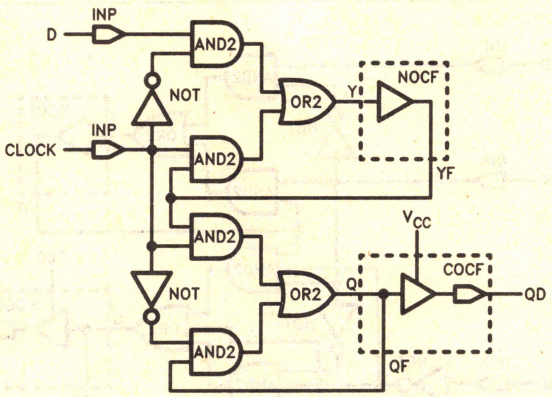


Figure 3. Combinational Logic Implementation of a D Flip-Flop

Preset and clear can be added into the equations as well:

```
Q0,QF = COCF (Q,VCC)
YF = NOCF (Y)

Y = D * !CLOCK + YF * CLOCK;
Q = YF * CLOCK * ! (CLEAR TERM) +
    (PRESET TERM) +
    QF * !CLOCK * ! (CLEAR TERM);
```

When the PRESET TERM is logically true, Q is asynchronously set to 1.

When the CLEAR TERM is logically true, Q is asynchronously cleared to 0.
The PRESET TERM takes priority over the CLEAR TERM.
This schematic is shown in Figure 4.

Due to the nature of the design, input delays plus array delays plus feedback delays must be added and used to determine a maximum operating frequency. In this example, tIN + tTAD + tCF + tAD = 113 ns for a −65 5C121, leaving a maximum frequency of 8.8 MHz.

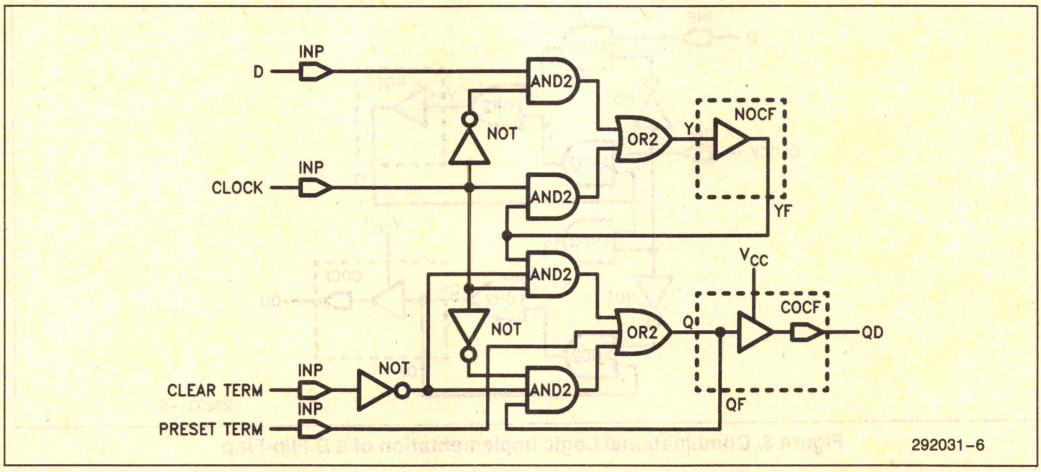


Figure 4. D Flip-Flop with Added Preset and Clear Terms

Other useful workarounds involve D registers and logic in constructing RS, JK and T flip-flops, for use in EPLDs not supporting these configurations. The RS flip-flop is simply the RS latch discussed earlier coupled to registered feedback.

```
Q0,QF = RORF (Q,CLOCK,GND,GND;VCC)
Q = S + QF * !R;
```

Normally, S and R will remain high. When S is brought low, Q0 will become 1 on the next clock trigger edge. When R is brought low, Q0 will become 0 on the next clock trigger edge. The schematic is given in Figure 5.

The JK flip-flop is another useful and easily implemented register:

```
Q0,QF = RORF (Q,CLOCK,GND,GND,VCC)
Q = J * !QF + !K * QF
```

When J = K = 1, QO toggles to opposite state on next clock trigger. When J = K = 0, QO remains the same. When J does not equal K, QO will follow J on next clock trigger. The schematic is shown in Figure 6.

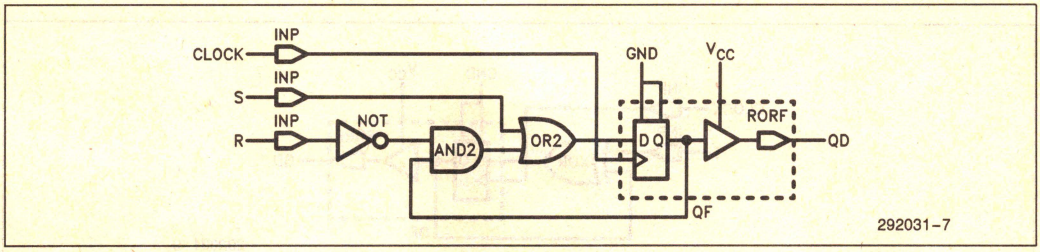


Figure 5. EPLD Implementation of an RS Flip-Flop

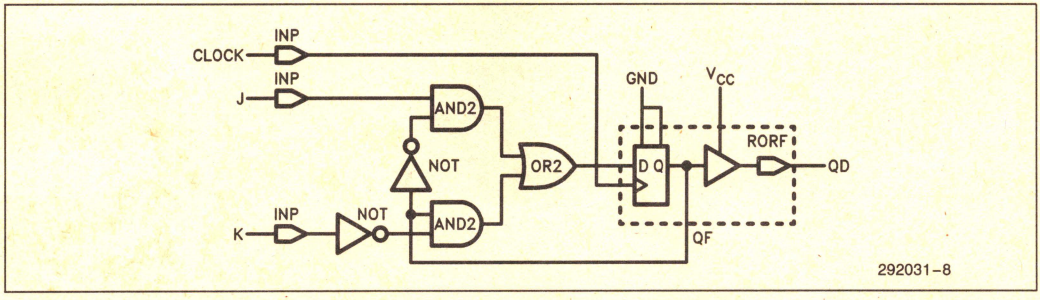


Figure 6. EPLD Implementation of a JK Flip-Flop

The T flip-flop is also easily constructed:

```
Q0,QF = RORF (Q,CLOCK,GND,GND,VCC)
Q = T * ! QF + ! T * QF;
```

When T is high, Q0 will toggle to opposite state on next trigger. When T is low, Q0 will remain the same. Figure 7 shows the T flip-flop design schematic.

Each of these designs uses a minimum number of p-terms; adding p-terms is possible to the limit of the macrocell being used. It is possible to substitute an entire logical expression for each input listed (except register clock), as long as the minimized logic equations resulting do not exceed the macrocells p-term count.

For example, consider using the J-K register. Setting J = A * B * C + D and setting K = E * !F * !G + H + I then the minimized p-term count will expand from two p-terms to five p-terms, which would still be okay within a macrocell with more than five p-terms.

Using logic gates and combinational or registered feedback, one can easily implement many types of latches and registers. Regardless of the EPLD type, there exists the resources to implement any of the discussed circuitry.

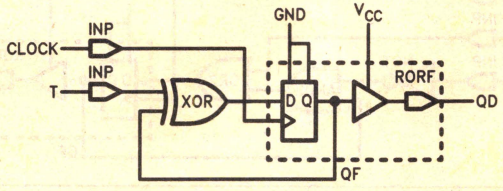


Figure 7. Implementation of a T Flip-Flop

intel

APPLICATION BRIEF

AB-18

October 1987

TTL Macro Library Listing for EPLD Designs

PROGRAMMABLE LOGIC APPLICATIONS
INTEL CORPORATION

Order Number: 292037-002

TTL Macros

The following is a partial list of TTL macros that are available through the Intel EPLD customer hot line.

These macros are used with the SCHEMA II-PLD schematic capture package. They can also be used in ADFs (Advanced Design Files) created using a text editor.

THIS LIST REPRESENTS VERSION 3.4 OF THE TTL MACRO LIBRARY. FUTURE VERSIONS ARE SUBJECT TO CHANGE.

SSI GATES

7400	2 Input NAND
7402	2 Input NOR
7404	1 Input INVERTER
7408	2 Input AND
7410	3 Input NAND
7411	3 Input AND
7420	4 Input NAND
7421	4 Input AND
7427	3 Input NOR
7430	8 Input NAND
7432	2 Input OR
7486	2 Input XOR

MSI FUNCTIONS

Decoders/Demultiplexers

7442	(10) BCD to Decimal
7444	(10) Excess-3-Gray to Decimal
7447X	(7) BCD to 7-Segment—Active Low Output
7449	(7) BCD to 7-Segment—Active High Output
74138	(8) 1-of-8 Decoder
74139	(4) Single 1-of-4 Decoder
74145	(10) BCD to Decimal
74154	(16) 1-of-16 Decoder
74155	(8) Dual 1-of-4
74156	(8) Dual 1-of-4

Multiplexers

74151	(2) 8-to-1
74153	(2) Dual 4-to-1—Active High Output
74157	(4) Quad 2-to-1—Active High Output
74158	(4) Quad 2-to-1—Active Low Output
74253	(2) Dual 4-to-1—Three-State Output
74257X	(4) Quad 2-to-1—Active High, Three-State Output
74258X	(4) Quad 2-to-1—Active Low, Three-State Output
74298XA	(4) Quad 2-to-1—Active High with Storage
74298XB	(4) Quad 2-to-1—Active High with Storage
74352	(2) Dual 4-to-1—Active Low Output

Counters

		Type	Clear	Load	Clk	Extras
7490XD	(4)	BCD Decade	S	9	R	
7490XQ	(4)	Bi-Quinary	S	9	R	
74160	(5)	BCD Decade	A	S	R	RCO
74161	(5)	4-Bit Binary	A	S	R	RCO
74162	(5)	BCD Decade	S	S	R	RCO
74163	(5)	4-Bit Binary	S	S	R	RCO
74168	(5)	BCD Decade	—	S	R	U/D, RCO
74169	(5)	4-Bit Binary	—	S	R	U/D, RCO
74176XD	(4)	BCD Decade	A	S	R	
74176XQ	(4)	Bi-Quinary	A	S	R	
74177X	(4)	4-Bit Binary	A	S	R	
74190XA	(6)	BCD Decade	—	S	R	U/D, RCO, MM
74190XB	(6)	BCD Decade	—	S	R	U/D, RCO, MM
74191XA	(7)	4-Bit Binary	—	S	R	U/D, RCO, MM
74290XD	(4)	BCD Decade	S	9	R	
74290XQ	(4)	Bi-Quinary	S	9	R	
74390X	(4)	Bi-Quinary/BCD	A	—	F	
74393XA	(4)	4-Bit Binary	A	—	F	
74393XB	(4)	4-Bit Binary	A	—	F	

S = Synchronous
A = Asynchronous
9 = Synchronous Set-to-9

R = Rising-Edge Triggered
F = Falling-Edge Triggered

U/D = Up/Down
RCO = Ripple Carry Output
MM = Max/Min Output

Single Flip-Flops

7472XA	(2)	AND-Gated JK Master/Slave
7472XB	(2)	AND-Gated JK Master/Slave
7473X	(2)	JK with Clear
7474X	(2)	D with Preset and Clear
74112XA	(3)	JK with Preset and Clear
74112XB	(2)	JK with Clear

Latches

7475X	(8)	4-Bit Bistable
7477X	(4)	Quad D-Type
74259XA	(8)	Octal Addressable D-Type
74259XB	(8)	Octal Addressable D-Type
74373X	(8)	Octal D-Type

Multiple Flip-Flops (Registers)

74174X	(6)	Hex D
74175X	(8)	Quad D with Q and /Q
74273X	(8)	Octal D
74377	(8)	Octal D with Common Enable
74378	(6)	Hex D

Shift Registers

7491	(8)	8-Bit—Serial-In, Serial-Out
7495XA	(4)	4-Bit—Serial-In/Parallel-In, Parallel-Out
7495XB	(4)	4-Bit—Serial-In/Parallel-In, Parallel-Out
7495XC	(4)	4-Bit—Serial-In/Parallel-In, Parallel-Out
7496X	(5)	5-Bit—Serial-In/Parallel-In, Parallel-Out
74164	(8)	8-Bit—Serial-In, Parallel-Out
74165X	(9)	8-Bit—Serial-In/Parallel-In, Serial-Out
74194	(4)	4-Bit Bi-Directional—Serial-In/Parallel-In, Parallel-Out
74395XA	(5)	4-Bit Cascadable—Serial-In/Parallel-In, Parallel-Out
74395XA	(5)	4-Bit Cascadable—Serial-In/Parallel-In, Parallel-Out

Miscellaneous

7482X	(4)	2-Bit Adder
7483X	(8)	4-Bit Adder
7485X	(7)	4-Bit Magnitude Comparator
7487	(4)	4-Bit True/Complement Element
74143X	(17)	4-Bit Counter; 4-Bit Latch; 7 Segment Decoder
74180X	(4)	8-Bit Parity Generator/Checker
74180XA	(4)	8-Bit Parity Generator/Checker
74182	(5)	Look-Ahead Carry Generator
74183	(2)	Single-Bit Full Adder with Carry/Save
74280X	(5)	9-Bit Odd/Even Parity Generator/Checker

DEMORGAN EQUIVALENTS (BUBBLE GATES)

	Bubble AND (NOR)	Bubble NAND (OR)	Bubble NOR (AND)	Bubble OR (NAND)
2 Input	BAND2	BNAND2	BNOR 2	BOR2
3 Input	BAND3	BNAND3	BNOR 3	BOR3
4 Input	BAND4	BNAND4	BNOR 4	BOR4
6 Input	BAND6	BNAND6	BNOR 6	BOR6
8 Input	BAND8	BNAND8	BNOR 8	BOR8
12 Input	BAND12	BNAND12	BNOR 12	BOR12

INPUT/OUTPUT MACROS

INPUT	N/A	Generates Input Pin and Node in ADF
OUTPUT	(1)	Generates Enabled Output Buffer in ADF
OUTP	(1)	Output Pin (Used in SCHEMA II-PLD)
74125	(1)	Single Three-State Output, Active Low Enable
74126	(1)	Single Three-State Output, Active High Enable

NOTES:

1. All TTL macros duplicate TTL function only. They DO NOT DUPLICATE performance characteristics such as open-collector, totem-pole, or high-drive output.
2. Any TTL macros which deviate in some way from standard TTL function are denoted with an appended "X" (see device .DOC file for details). Appended "D"s and "Q"s indicate counters configured to Decimal or bi-Quinary mode; appended "A"s and "B"s indicate a macro configured for a family of EPLD devices (e.g. 5C060, 5C090, 5C180).
3. The (#) indicates the maximum number of EPLD macrocells consumed if all outputs are used. If an output is not used, the macro compression phase of the Macro Expander will remove the signal unless it is used as feedback inside the macro definition.
4. /Q's should be avoided as pin outputs if possible. The EPLD is structured such that the Q is readily available as a pin output and both the Q and /Q are readily available as feedbacks. Using /Q as a pin output, however, requires an extra macrocell and adds to the propagation delay.

intel

APPLICATION NOTE

AP-271

April 1986

Applying The 5C121 Architecture

JIM DONNELL
PROGRAMMABLE LOGIC APPLICATIONS
INTEL CORPORATION

Order Number: 292008-001

INTRODUCTION

Intel's 5C121 Erasable Programmable Logic Device represents a new breed in the world of programmable logic. With gate densities approaching those of gate arrays and a reconfigurable architecture, the logic designer is freed from choosing between scores of generic programmable logic to perhaps find an acceptable match for his or her design needs. Adding to the list of benefits is the fact that the 5C121 is erasable. Now sections of the design can actually be programmed and tested in the device — without sacrificing a part to the circular file. In addition, there is no longer a need to generate test vectors to qualify the programming of the parts. EPLDs are erasable and therefore 100% testable at the factory.

OBJECTIVE

The purpose of this application note is to demonstrate the architectural options of the 5C121 by designing a digital crosspoint switch. Conceptually, a digital crosspoint switch switches data from any input to any output. Figure 1 shows a block diagram of a bytewide crosspoint switch.

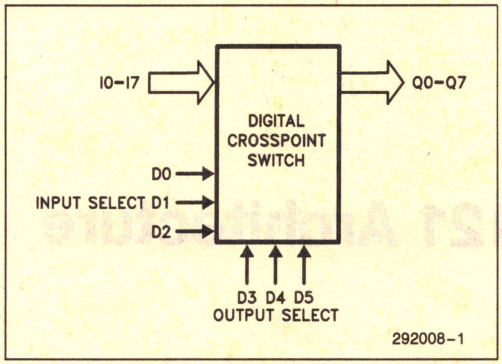


Figure 1. Functional Diagram of a Digital Crosspoint Switch

This design will employ features such as: registered output with registered feedback, combinational feedback, input latches, buried registers, and dual clock options. The digital crosspoint switch in this design can route data from one of eight inputs to one of eight outputs in a single clock cycle. Options for holding the deselected outputs at previous levels, latching inputs, and fitting considerations are explored.

THE BASIC ARCHITECTURE

The 5C121 contains 28 Macrocells, 12 dedicated inputs, 24 programmable I/O lines, and two clocks input pins. Inputs may be flow through, or latched on the rising or falling edge of either clock. Output options include registered or combinational output. In addition, each output may be fed back into the array in both the true and complement version. For a more complete description of the 5C121 architecture the reader is referred to the 5C121 data sheet.

COMBINATIONAL FEEDBACK

Feedback in logic designs is used for a variety of reasons. Combinational feedback in the 5C121 is often used to reduce the number of product terms feeding one Macrocell. Though the 5C121 has Macrocells that can accept up to 16 product terms, all Macrocells are not that wide.

Let's look at an example. Equation 1 represents one of the eight Boolean expressions necessary to implement a digital crosspoint switch. Logically, this expression selects one of eight input signals (I0–I7), and routes that signal to Q0. Data bits D0, D1, and D2 select one of the eight input lines. In this case, data bits !D3, !D4, and !D5 select output Q0. (The exclamation point is used to indicate a logical complement of the signal.) Equations for Q1 through Q7 are very similar and will be discussed later.

```
Q0 = ( I0 × !D2 × !D1 × !D0
     + I1 × !D2 × !D1 × D0
     + I2 × !D2 × D1 × !D0
     + I3 × !D2 × D1 × D0
     + I4 × D2 × !D1 × !D0
     + I5 × D2 × !D1 × D0
     + I6 × D2 × D1 × !D0
     + I7 × D2 × D1 × D0 ) × !D5 × !D4 × !D3;   (1)

SELECTEQ = I0 × !D2 × !D1 × !D0
         + I1 × !D2 × !D1 × D0
         + I2 × !D2 × D1 × !D0
         + I3 × !D2 × D1 × D0
         + I4 × D2 × !D1 × !D0
         + I5 × D2 × !D1 × D0
         + I6 × D2 × D1 × !D0
         + I7 × D2 × D1 × D0;                    (2)
```

Equation 2 contains the terms that will be common to all eight output equations. Both equations in this case contain eight product terms. By treating equation 2 as one common signal and routing that signal through combinational feedback, we can reduce the number of product terms in equations Q0 thru Q7 to one p-term each. The advantage is that the outputs can now be placed in any of the 24 I/O Macrocells available in the 5C121. In addition, the 5C121 contains four buried registers. (Buried registers have no output and are used solely for feedback.) If a buried register is available, iPLDs (Intel's Programmable Logic Development System) will automatically assign the No Output — Combinational Feedback function to a buried register. This increases the flexibility for pin assignments and makes

COMBINATIONAL FEEDBACK
(Continued)

p-terms available in case a design change is needed. Equations 3 thru 10 reflect this improvement.

Q0 = SELECTEQ × !D5 × !D4 × !D3; (3)

Q1 = SELECTEQ × !D5 × !D4 × D3; (4)

Q2 = SELECTEQ × !D5 × D4 × !D3; (5)

Q3 = SELECTEQ × !D5 × D4 × D3; (6)

Q4 = SELECTEQ × D5 × !D4 × !D3; (7)

Q5 = SELECTEQ × D5 × !D4 × D3; (8)

Q6 = SELECTEQ × D5 × D4 × !D3; (9)

Q7 = SELECTEQ × D5 × D4 × D3; (10)

REGISTERED FEEDBACK

Registered feedback is also employed in a variety of applications such as counters and state machines. In this particular example, the registered feedback signal can be used to hold the deselected outputs of the switch at their previous level until that output is selected again. This is accomplished by simply "ANDing" the feedback signal with the inversion of the output select signal. The result is then "ORed" with the equation for the given output. Holding the previous output might be useful in control applications or when interfacing to slow peripherals. Equations 11 thru 18 are the result.

Q0 = SELECTEQ × !D5 × !D4 × !D3 + !(D5 × !D4 × !D3) × Q0—fdbk; (11)

Q1 = SELECTEQ × !D5 × !D4 × D3 + !(!D5 × !D4 × D3) × Q1—fdbk; (12)

Q2 = SELECTEQ × !D5 × D4 × !D3 + !(!D5 × D4 × !D3) × Q2—fdbk; (13)

Q3 = SELECTEQ × !D5 × D4 × D3 + !(!D5 × D4 × D3) × Q3—fdbk; (14)

Q4 = SELECTEQ × D5 × !D4 × D3 + !(D5 × !D4 × !D3) × Q4—fdbk; (15)

Q5 = SELECTEQ × D5 × !D4 × D3 + !(D5 × !D4 × D3) × Q5—fdbk; (16)

Q6 = SELECTEQ × D5 × D4 × !D3 + !(D5 × D4 × !D3) × Q6—fdbk; (17)

Q7 = SELECTEQ × D5 × DR × D3 + !(D5 × D4 × DE) × Q7—fdbk; (18)

Equations 11 thru 18 are all that are necessary to implement a digital crosspoint switch with the output hold feature. Each equation contains only four product terms when written in the expanded form and could therefore fit into any Macrocell in the 5C121. The appendix contains the report and ADF files generated by the iPLDs software.

TIMING ANALYSIS

Figure 2 shows the internal delay paths associated with this design in the 5C121. The frequency at which the 5C121 may be clocked can be determined by examining the internal delay elements of the 5C121. These include the input delay (Tin), two array delays (Tad), and the combinational feedback delay (Tcf). Table 1 gives the simulation data for each of these paths in a 5C121-50.

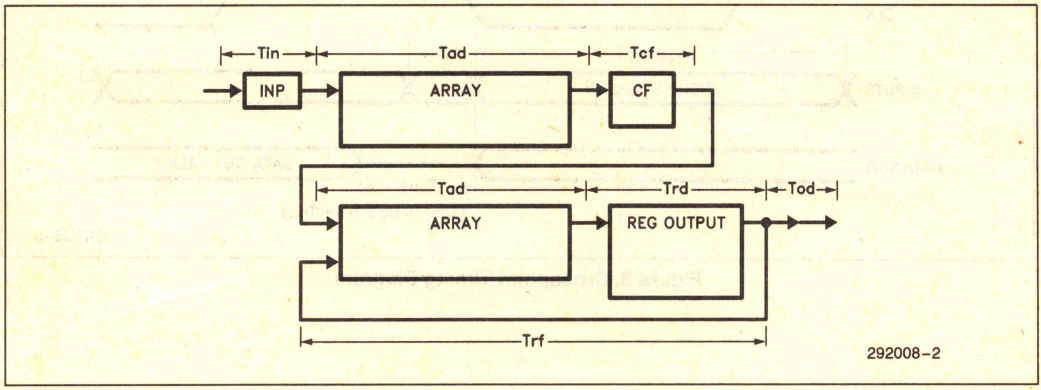


Figure 2. Crosspoint Delay Path

TIMING ANALYSIS (Continued)

Table 1. 5C121-50 Simulation Data

Model Parameter	Delay (ns)
Tad	38
Trd	7
Tod	8
Tin	10
Tic	8
Trf	5
Tcf	5

The sum of the delays before the register input equal the set-up time Tsu with reference to the internal clock. By substracting the input clock delay Tic we shift the reference to the external clock pin. The set-up time with reference to external signals is shown in equation 19. Inverting this signal yields the maximum clock frequency, fmax. The maximum clock frequency is shown in equation 20.

$$Tsu = Tin + 2Tad + Tcf - Tic; \quad (19)$$

$$fmax = 1\ Tsu \quad (20)$$

Therefore, this configuration of the 5C121-50 could be clocked at 10 MHz, allowing a data transfer rate of 10 Mbits/second. By paralleling six 5C121s together, eight bits could be switched per cycle. Figure 3 shows the timing diagram for this configuration of the 5C121 digital crosspoint switch. Included in the appendix is the Advanced Design File (ADF), Logic Equation File (LEF), and Utilization report generated by Intel's Programmable Logic Software (iPLS) for this design.

INPUT LATCHES

One point must be raised about Figure 3. Notice that the time allowed for external data set-up is only 17 ns. Therefore, 17 ns after the rising edge of the clock, data must be stable and remain stable at the input pins until the next clock pulse. In most systems this would be a very stringent requirement. Fortunately the 5C121 has the ability to latch the data at the input pins with 7475 type transparent latches. Employing this feature eases the data set-up requirement as shown in Figure 4.

SUMMARY

The flexible architecture of the 5C121 gives the designer a variety of options for input and output configurations. Inputs may be latched to ease system timing requirements. Outputs may be clocked for synchronous systems or fed directly out as asynchronous signals. Feedback can be used to reduce product term requirements, to save present state information for state machines and counters, or simply to hold deselected outputs as shown in this example. Imagine the possibilities.

J. R. Donnell
PLDO Applications

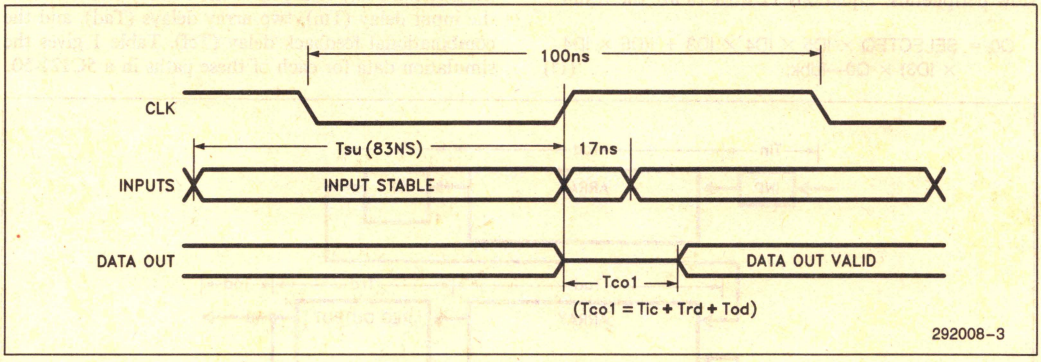


Figure 3. Crosspoint Timing Diagram

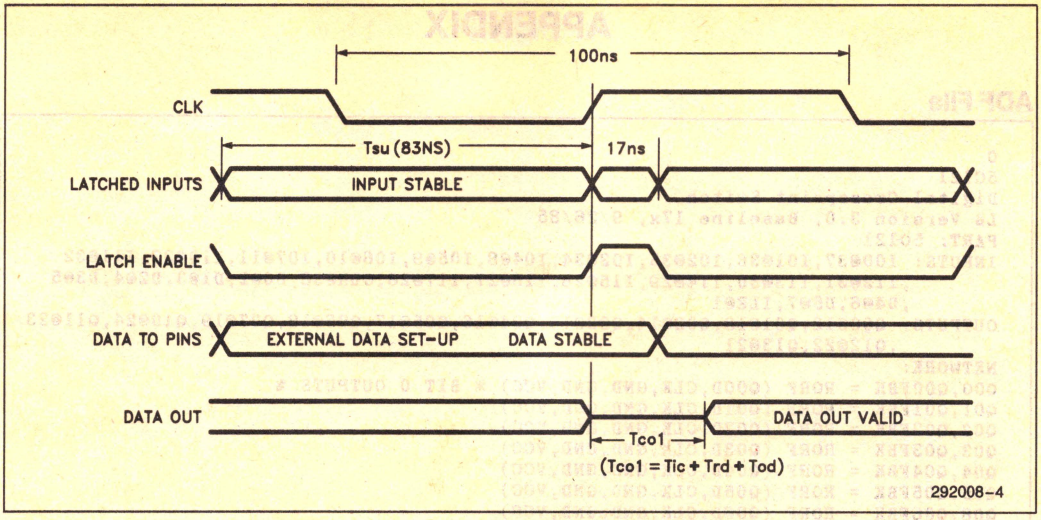


Figure 4. Crosspoint Timing Diagram with Input Latches

APPENDIX

ADF File

```
0
5C121
Digital Crosspoint Switch
LB Version 3.0, Baseline 17x, 9/26/85
PART: 5C121
INPUTS: I00@37,I01@36,I02@35,I03@34,I04@8,I05@9,I06@10,I07@11,I10@33,I11@32
       ,I12@31,I13@30,I14@29,I15@28,I16@27,I17@26,CLK@38,D0@2,D1@3,D2@4,D3@5
       ,D4@6,D5@7,ILE@1
OUTPUTS: Q00@12,Q01@13,Q02@14,Q03@15,Q04@16,Q05@17,Q06@18,Q07@19,Q10@24,Q11@23
        ,Q12@22,Q13@21
NETWORK:
Q00,Q00FBK = RORF (Q00D,CLK,GND,GND,VCC) % BIT 0 OUTPUTS %
Q01,Q01FBK = RORF (Q01D,CLK,GND,GND,VCC)
Q02,Q02FBK = RORF (Q02D,CLK,GND,GND,VCC)
Q03,Q03FBK = RORF (Q03D,CLK,GND,GND,VCC)
Q04,Q04FBK = RORF (Q04D,CLK,GND,GND,VCC)
Q05,Q05FBK = RORF (Q05D,CLK,GND,GND,VCC)
Q06,Q06FBK = RORF (Q06D,CLK,GND,GND,VCC)
Q07,Q07FBK = RORF (Q07D,CLK,GND,GND,VCC)
Q10,Q10FBK = RORF (Q10D,CLK,GND,GND,VCC) % 4 OF THE 8, BIT 0 OUTPUTS%
Q11,Q11FBK = RORF (Q11D,CLK,GND,GND,VCC)
Q12,Q12FBK = RORF (Q12D,CLK,GND,GND,VCC)
Q13,Q13FBK = RORF (Q13D,CLK,GND,GND,VCC)
CLK = INP (CLK)
D5 = LINP (D5,ILE) % OUTPUT SELECT CONTROL BITS %
ILE = INP (ILE)
D4 = LINP (D4,ILE)
D3 = LINP (D3,ILE)
D2 = LINP (D2,ILE) % INPUT SELECT CONTROL BITS %
D1 = LINP (D1,ILE)
D0 = LINP (D0,ILE)
I00 = LINP (I00,ILE)
I01 = LINP (I01,ILE)
I02 = LINP (I02,ILE)
I03 = LINP (I03,ILE)
I04 = LINP (I04,ILE)
I05 = LINP (I05,ILE)
I06 = LINP (I06,ILE)
I07 = LINP (I07,ILE)
I10 = LINP (I10,ILE) % INPUTS FOR BIT 1 SWITCH %
I11 = LINP (I11,ILE)
I12 = LINP (I12,ILE)
I13 = LINP (I13,ILE)
I14 = LINP (I14,ILE)
I15 = LINP (I15,ILE)
I16 = LINP (I16,ILE)
I17 = LINP (I17,ILE)
SELECTEQ0F = NOCF (SELECTEQ0)
SELECTEQ1F = NOCF (SELECTEQ1)
EQUATIONS:
Q00D = SELECTEQ0F*!D5*!D4*!D3
        + !(!D5*!D4*!D3)*Q00FBK;
Q01D = SELECTEQ0F*!D5*!D4* D3
        + !(!D5*!D4* D3)*Q01FBK;
Q02D = SELECTEQ0F*!D5* D4*!D3
        + !(!D5* D4*!D3)*Q02FBK;
Q03D = SELECTEQ0F*!D5* D4* D3
        + !(!D5* D4* D3)*Q03FBK;
Q04D = SELECTEQ0F* D5*!D4*!D3
        + !( D5*!D4*!D3)*Q04FBK;
Q05D = SELECTEQ0F* D5*!D4* D3
```

292008-5

ADF File (Continued)

```
        + !( D5*!D4* D3)*Q05FBK;
Q06D = SELECTEQ0F* D5* D4*!D3
        + !( D5* D4*!D3)*Q06FBK;
Q07D = SELECTEQ0F* D5* D4* D3
        + !( D5* D4* D3)*Q07FBK;
Q10D = SELECTEQ1F*!D5*!D4*!D3
        + !(!D5*!D4*!D3)*Q10FBK;
Q11D = SELECTEQ1F*!D5*!D4* D3
        + !(!D5*!D4* D3)*Q11FBK;
Q12D = SELECTEQ1F*!D5* D4*!D3
        + !(!D5* D4*!D3)*Q12FBK;
Q13D = SELECTEQ1F*!D5* D4* D3
        + !(!D5* D4* D3)*Q13FBK;
SELECTEQ0 = I00*!D2*!D1*!D0      % COMMON EQUATION FOR BIT 0 %
        + I01*!D2*!D1*D0
        + I02*!D2*D1*!D0
        + I03*!D2*D1*D0
        + I04*D2*!D1*!D0
        + I05*D2*!D1*D0
        + I06*D2*D1*!D0
        + I07*D2*D1*D0;
SELECTEQ1 = I10*!D2*!D1*!D0      % COMMON EQUATION FOR BIT 1 %
        + I11*!D2*!D1*D0
        + I12*!D2*D1*!D0
        + I13*!D2*D1*D0
        + I14*D2*!D1*!D0
        + I15*D2*!D1*D0
        + I16*D2*D1*!D0
        + I17*D2*D1*D0;
END$
```

292008-6

LEF File

```
JR Donnell
Intel
January 24, 1986

0
5C121
Digital Crosspoint Switch
LB Version 3.0, Baseline 17x, 9/26/85
PART:
        5C121
INPUTS:
        I00@37, I01@36, I02@35, I03@34, I04@8, I05@9, I06@10, I07@11, I10@33,
        I11@32, I12@31, I13@30, I14@29, I15@28, I16@27, I17@26, CLK@38, D0@2,
        D1@3, D2@4, D3@5, D4@6, D5@7, ILE@1
OUTPUTS:
        Q00@12, Q01@13, Q02@14, Q03@15, Q04@16, Q05@17, Q06@18, Q07@19, Q10@24,
        Q11@23, Q12@22, Q13@21
NETWORK:
        CLK = INP(CLK)
        ILE = INP(ILE)
        I00 = LINP(I00, ILE)
        I01 = LINP(I01, ILE)
        I02 = LINP(I02, ILE)
        I03 = LINP(I03, ILE)
        I04 = LINP(I04, ILE)
        I05 = LINP(I05, ILE)
        I06 = LINP(I06, ILE)
        I07 = LINP(I07, ILE)
        I10 = LINP(I10, ILE)
        I11 = LINP(I11, ILE)
        I12 = LINP(I12, ILE)
        I13 = LINP(I13, ILE)
        I14 = LINP(I14, ILE)
        I15 = LINP(I15, ILE)
        I16 = LINP(I16, ILE)
        I17 = LINP(I17, ILE)
        D0 = LINP(D0, ILE)
        D1 = LINP(D1, ILE)
        D2 = LINP(D2, ILE)
        D3 = LINP(D3, ILE)
        D4 = LINP(D4, ILE)
        D5 = LINP(D5, ILE)
        Q00, Q00FBK = RORF(Q00D, CLK, GND, GND, VCC)
        Q01, Q01FBK = RORF(Q01D, CLK, GND, GND, VCC)
        Q02, Q02FBK = RORF(Q02D, CLK, GND, GND, VCC)
        Q03, Q03FBK = RORF(Q03D, CLK, GND, GND, VCC)
        Q04, Q04FBK = RORF(Q04D, CLK, GND, GND, VCC)
        Q05, Q05FBK = RORF(Q05D, CLK, GND, GND, VCC)
        Q06, Q06FBK = RORF(Q06D, CLK, GND, GND, VCC)
        Q07, Q07FBK = RORF(Q07D, CLK, GND, GND, VCC)
        Q10, Q10FBK = RORF(Q10D, CLK, GND, GND, VCC)
        Q11, Q11FBK = RORF(Q11D, CLK, GND, GND, VCC)
        Q12, Q12FBK = RORF(Q12D, CLK, GND, GND, VCC)
        Q13, Q13FBK = RORF(Q13D, CLK, GND, GND, VCC)
        SELECTEQ0F = NOCF(SELECTEQ0)
        SELECTEQ1F = NOCF(SELECTEQ1)
EQUATIONS:
        SELECTEQ1 = I10 * D2' * D1' * D0'
                  + D2 * D1' * D0' * I14
                  + D2' * D1 * D0' * I12
                  + D2' * D1' * D0 * I11
                  + D2 * D1 * D0' * I16
                  + D2 * D1' * D0 * I15
                  + D2' * D1 * D0 * I13
```

292008-12

LEF File (Continued)

```
                + D2 * D1 * D0 * I17;

SELECTEQ0 = I00 * D2' * D1' * D0'
          + D2 * D1' * D0' * I04
          + D2' * D1 * D0' * I02
          + D2' * D1' * D0 * I01
          + D2 * D1 * D0' * I06
          + D2 * D1' * D0 * I05
          + D2' * D1 * D0 * I03
          + D2 * D1 * D0 * I07;

Q13D = D3' * Q13FBK
     + D4' * Q13FBK
     + D5 * Q13FBK
     + SELECTEQ1F * D5' * D4 * D3;

Q12D = D4' * Q12FBK
     + D3 * Q12FBK
     + D5 * Q12FBK
     + SELECTEQ1F * D5' * D4 * D3';

Q11D = D3' * Q11FBK
     + D4 * Q11FBK
     + D5 * Q11FBK
     + SELECTEQ1F * D5' * D4' * D3;

Q10D = D3 * Q10FBK
     + D4 * Q10FBK
     + D5 * Q10FBK
     + SELECTEQ1F * D5' * D4' * D3';

Q07D = D3' * Q07FBK
     + D4' * Q07FBK
     + D5' * Q07FBK
     + SELECTEQ0F * D5 * D4 * D3;

Q06D = D4' * Q06FBK
     + D5' * Q06FBK
     + D3 * Q06FBK
     + SELECTEQ0F * D5 * D4 * D3';

Q05D = D3' * Q05FBK
     + D5' * Q05FBK
     + D4 * Q05FBK
     + SELECTEQ0F * D5 * D4' * D3;

Q04D = D5' * Q04FBK
     + D3 * Q04FBK
     + D4 * Q04FBK
     + SELECTEQ0F * D5 * D4' * D3';

Q03D = D3' * Q03FBK
     + D4' * Q03FBK
     + D5 * Q03FBK
     + SELECTEQ0F * D5' * D4 * D3;

Q02D = D4' * Q02FBK
     + D3 * Q02FBK
     + D5 * Q02FBK
     + SELECTEQ0F * D5' * D4 * D3';

Q01D = D3' * Q01FBK
     + D4 * Q01FBK
     + D5 * Q01FBK
     + SELECTEQ0F * D5' * D4' * D3;
```

292008-13

```
Q00D = D3 * Q00FBK
     + D4 * Q00FBK
     + D5 * Q00FBK
     + SELECTEQ0F * D5' * D4' * D3';

END$
```

292008-14

RPT File

```
Logic Optimizing Compiler Utilization Report

 ***** Design implemented successfully

JR Donnell
Intel
January 24, 1986

0
5C121
Digital Crosspoint Switch
LB Version 3.0, Baseline 17x, 9/26/85

                5C121
              - - - - -
        ILE -| 1   40|- Vcc
         D0 -| 2   39|- Vcc
         D1 -| 3   38|- CLK
         D2 -| 4   37|- I00
         D3 -| 5   36|- I01
         D4 -| 6   35|- I02
         D5 -| 7   34|- I03
        I04 -| 8   33|- I10
        I05 -| 9   32|- I11
        I06 -|10   31|- I12
        I07 -|11   30|- I13
        Q00 -|12   29|- I14
        Q01 -|13   28|- I15
        Q02 -|14   27|- I16
        Q03 -|15   26|- I17
        Q04 -|16   25|- GND
        Q05 -|17   24|- Q10
        Q06 -|18   23|- Q11
        Q07 -|19   22|- Q12
        GND -|20   21|- Q13
              - - - - -


**INPUTS**

                                                                  |          Feeds:
      Name  Pin  Resource   MCell #     PTerms                     |  MCells     OE  Clear  Clock

       ILE    1       INP         -          -                     |        -      -      -  Latch

        D0    2      LINP         -          -                     |       13      -      -      -
                                                                  |       15

        D1    3      LINP         -          -                     |       13      -      -      -
                                                                  |       15

        D2    4      LINP         -          -                     |       13      -      -      -
                                                                  |       15

        D3    5      LINP         -          -                     |        9      -      -      -
                                                                  |       10
                                                                  |       11
                                                                  |       12
                                                                  |       17
                                                                  |       18
                                                                  |       19
                                                                  |       20
                                                                  |       21
```

292008-9

RPT File (Continued)

					22 23 24			
D4	6	LINP	-	-	9 10 11 12 17 18 19 20 21 22 23 24	-	-	-
D5	7	LINP	-	-	9 10 11 12 17 18 19 20 21 22 23 24	-	-	-
I04	8	LINP	28	0/ 4	15	-	-	-
I05	9	LINP	27	0/10	15	-	-	-
I06	10	LINP	26	0/ 8	15	-	-	-
I07	11	LINP	25	0/ 6	15	-	-	-
I17	26	LINP	7	0/10	13	-	-	-
I16	27	LINP	6	0/ 8	13	-	-	-
I15	28	LINP	5	0/ 6	13	-	-	-
I14	29	LINP	4	0/ 6	13	-	-	-
I13	30	LINP	3	0/ 8	13	-	-	-
I12	31	LINP	2	0/10	13	-	-	-
I11	32	LINP	1	0/ 4	13	-	-	-
I10	33	LINP	-	-	13	-	-	-
I03	34	LINP	-	-	15	-	-	-
I02	35	LINP	-	-	15	-	-	-
I01	36	LINP	-	-	15	-	-	-
I00	37	LINP	-	-	15	-	-	-
CLK	38	INP	-	-	-	-	-	Reg

292008-10

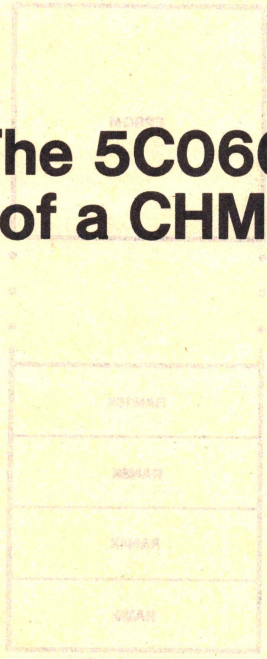
RPT File (Continued)

```
**OUTPUTS**

                                                                  Feeds:
Name  Pin  Resource  MCell #  PTerms  MCells  OE  Clear
Q00   12   RORF      24       4/ 6    24      -   -
Q01   13   RORF      23       4/ 8    23      -   -
Q02   14   RORF      22       4/10    22      -   -
Q03   15   RORF      21       4/ 4    21      -   -
Q04   16   RORF      20       4/12    20      -   -
Q05   17   RORF      19       4/ 4    19      -   -
Q06   18   RORF      18       4/ 8    18      -   -
Q07   19   RORF      17       4/ 8    17      -   -
Q13   21   RORF      12       4/ 8    12      -   -
Q12   22   RORF      11       4/ 8    11      -   -
Q11   23   RORF      10       4/ 4    10      -   -
Q10   24   RORF       9       4/12     9      -   -


**BURIED REGISTERS**

                                                                  Feeds:
Name  Pin  Resource  MCell #  PTerms  MCells  OE  Clear
      -    NOCF      13       8/ 8     9      -   -
                                       10
                                       11
                                       12
      -    NOCF      15       8/ 8    17      -   -
                                       18
                                       19
                                       20
                                       21
                                       22
                                       23
                                       24


**UNUSED RESOURCES**

Name  Pin  Resource  MCell  PTerms
-     25   -         8      4
-     NA   -         14     8
-     NA   -         16     8

**PART UTILIZATION**

97%   Pins
89%   MacroCells
30%   Pterms
```

292008-11

June 1986

The 5C060 Unification of a CHMOS System



J. R. DONNELL
PROGRAMMABLE LOGIC APPLICATIONS
INTEL CORPORATION

Order Number: 292009-003

INTRODUCTION

From an outside glance, the world of computers and microprocessors seems filled with dedicated ICs that fulfill a variety of system needs. Upon closer inspection we find that designers must still reach into their bag of random logic to link together all of the parts of the system. It seems a shame to stuff a board full of high powered peripherals and still have portions of that board wasted on decoders, latches, and other miscellaneous random logic.

True, programmable logic has been around a long time. But that logic is somewhat rigid in form, one time programmable, and can also double as space heaters. These devices are totally unacceptable for a CMOS system. What is needed is a flexible PLA architecture, erasability for prototyping, and CMOS for low power. In addition, for this particular application the device must perform from static operation to 10 MHz.

OBJECTIVE

This application note covers the design of three separate circuits for Intel's CHMOS Design Kit. The functions performed by the 5C060 are: Memory decoding, wait state generation, and the power down circuitry for the 80C88 system clock.

MEMORY DECODING

The system in question supports one 32K bank of EPROM memory, and four banks of 4K static RAM. Figure 1 shows the memory map of this system. Address lines A19, A13, and A12 will be used to decode the address space. PWR_DWN and S2_MIO serve as enables. In addition, to avoid data bus contention signals memory read (MRDC) and advanced memory write (AMWC) are decoded along with the address lines for RAM chip selects. This is necessary for devices without output enables (OE) on multiplexed address/data busses.

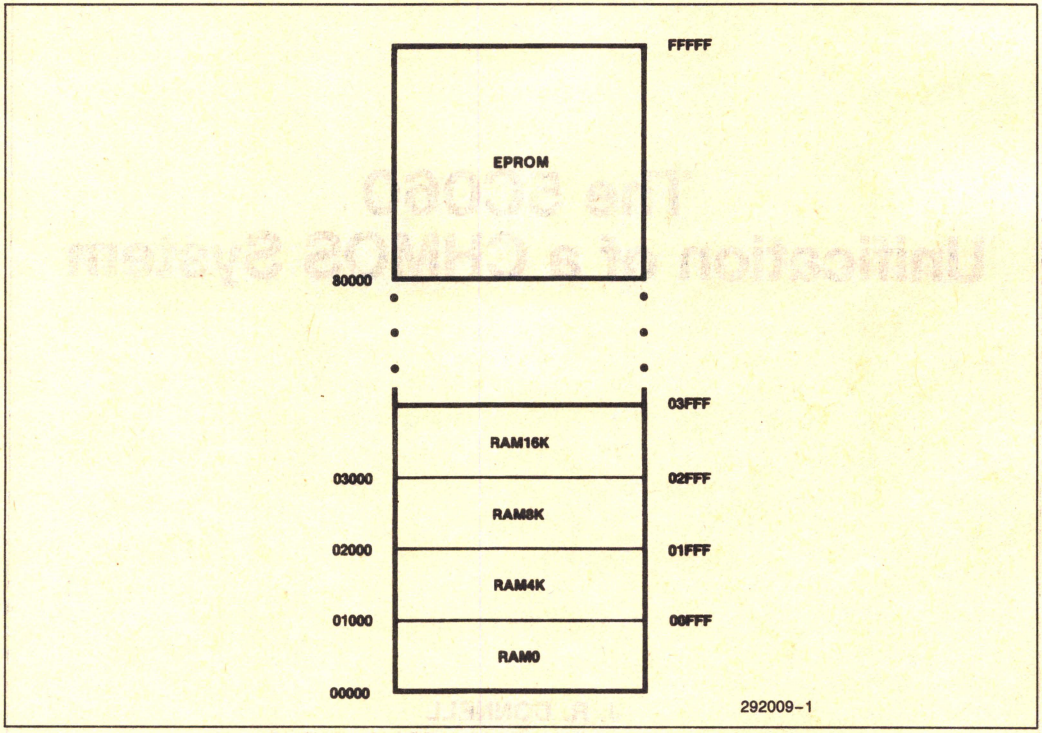


Figure 1. 80C88 Memory Map

Figure 2 shows a discrete implementation of the chip select decoding logic.

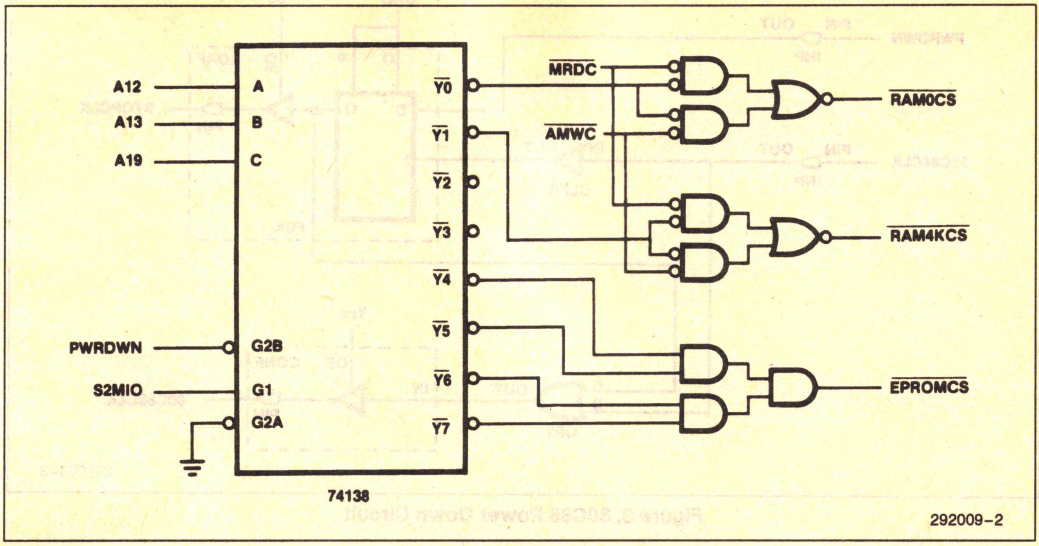


Figure 2. Discrete Decoding Logic Solution

Several options for entering this design are available through Intel's Programmable Logic Development System (iPLDS). (For a more complete description of iPLDS the reader is referred to the iPLDS data sheet.) The design entry vehicle chosen for this application note is the Logic Builder. (Logic Builder is an interactive netlist method of design entry especially suited to Boolean equation entry and entry from existing schematics.) Several reasons are behind this decision. First, the Logic Builder software is included in iPLDS. In addition, Logic Builder entry is very fast, the designer may choose either netlist entry or Boolean equations, and finally, the Logic Builder software makes additions and corrections of design very easy.

Using Logic Builder, the first step for this design is to determine the equations for the 3 to 8 decoder shown in Figure 2. These equations are simply the decoding of the address lines ANDed with the enable signal. Equations 0 thru 8 implement the decoding function of Figure 2.

```
/Y0 = /A19*/A13*/A12*ENABLE;          (0)
/Y1 = /A19*/A13*A12*ENABLE;           (1)
/Y2 = /A19*A13*/A12*ENABLE;           (2)
/Y3 = /A19*A13*A12*ENABLE;            (3)
/Y4 = A19*/A13*/A12*ENABLE;           (4)
/Y5 = A19*/A13*A12*ENABLE;            (5)
/Y6 = A19*A13*/A12*ENABLE;            (6)
/Y7 = A19*A13*A12*ENABLE;             (7)
ENABLE = /PWRDWN*S2MIO;               (8)
```

Armed with this knowledge it becomes trivial to enter the circuit of Figure 2 into Logic Builder. Included in the Appendix is the Advanced Design File (ADF) created by Logic Builder for this circuit (ADF-1). Typically the ADF would now be submitted to the Logic Optimizing Compiler (LOC) for Boolean minimization and design fitting. In this case we have used only a small portion if the logic available in the 5C060 so let us continue with the wait state generator and power down circuitry.

Power Down

Since this design is based on the 80C88 we can actually stop the system clock for extended periods of time and power back up as if nothing had occurred. The circuit to achieve this power down is shown in Figure 3.

As long as the PWRDWN signal is low the 82C84 clock output is OR'ed with a logical zero from the PWRDWN flip-flop. As a result the 82C84 drives the 80C88 system clock. If PWRDWN goes HIGH, the rising edge of the next 82C84 clock will set the output of the PWRDWN flip-flop HIGH inhibiting the fall of the next clock cycle. The 80C88 system clock will remain HIGH until PWRDWN goes LOW and the PWRDWN flip-flop is clocked from the 82C84 clock. Using this configuration we avoid partial clock cycles for the 80C88 system clock.

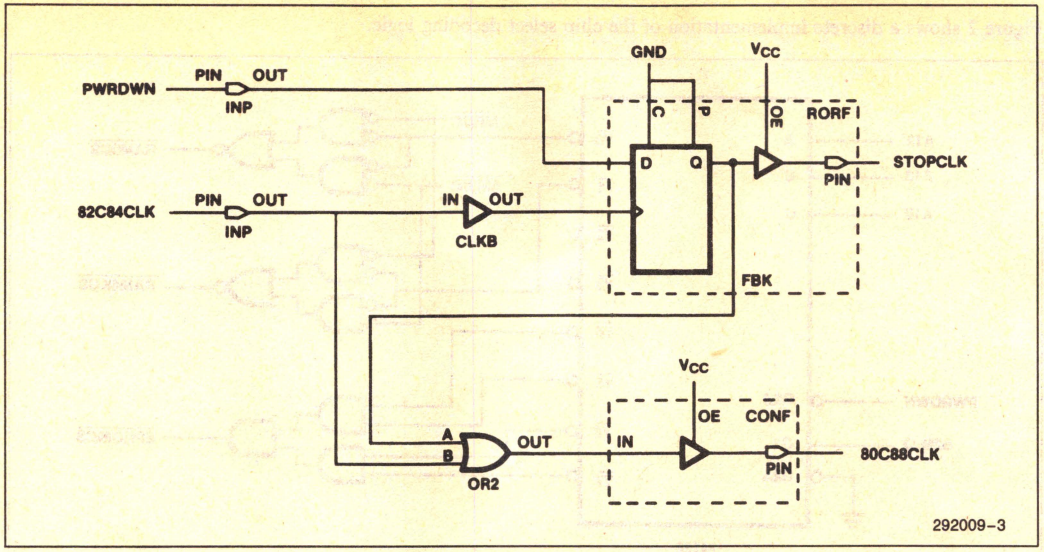


Figure 3. 80C88 Power Down Circuit

Again, entering this circuit into Logic Builder is trivial. In fact it can be added directly to the decoder circuit shown above. The ADF file for this addition is shown in the appendix under ADF-2.

Wait States

The majority of memory and peripheral devices which fail to operate at the maximum CPU frequency typically do not require more than one wait state. The circuit shown in Figure 4 is an example of a simple wait state generator. The circuit operation is as follows. Given that a memory location requiring a wait state has been selected, ALE in conjunction with /WAITCS will clear the flip-flop—driving the 82C84RDY line high low. The 82C84 samples the RDY line during T2 of the 80C88 bus cycle, and in this case detects a wait state. The rising edge of T2 then clocks the 82C84RDY line high thereby inserting only one wait state.

Once again, adding this circuit to the existing decoder and power down design is simple. The final ADF file is given in the appendix under ADF-3. Once the final design has been completed the ADF is submitted to the Logic Optimizing Compiler. LOC compiles the design, performs Boolean minimization, and fits the design into the target EPLD. In addition, LOC produces two files. The JEDEC programming file, the Logic Equation File (LEF), and the Utilization Report. These are also included in the appendix for each step in this design process.

LOC FILES

The JEDEC File

The JEDEC file is analogous to the object code file that is used to program EPROMs. This file is used by the Logic Programming Software (LPS) to program Intel's EPLDs.

The LEF File

The LEF file is an optional file produced by the compiler. The LEF file contains the minimized Boolean equations which resulted from the original ADF. Some interesting points can be raised concerning the LEF file. Looking at LEF-3, first recall that the EPROM chip select was a function of A19, A13, A12, and the enable signals. It turns out that after minimization the EPROM chip select depends only on A19 and the enable signals (/PWRDWN and S2MIO). This is shown in the LEF file. One other point, the initial wait state circuitry employed a JK flip-flop. The compiler automatically minimized this circuit into a D-type flip-flop with feedback achieving the same functionality.

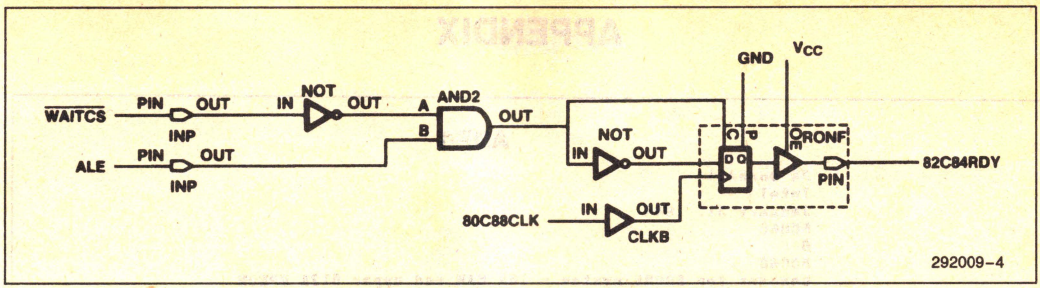


Figure 4. Single Wait State Generator for the 80C88

The Utilization Report

Finally, the Utilization Report contains the pin-out for the design, information about the architectural layout of the design, and a percent utilization for pins, macrocells, and product terms. Examining the utilization report for this design we find that two of the sixteen macrocells are still available. We could therefore add more functionality in the same 24 pin package. Possible additions would be more memory decoding, invalid memory detection, additional wait state generators, etc. One point should be raised: The circuitry designed in this applications note is relatively simple compared to the complex logic functions that could be implemented in the 5C060.

SUMMARY

The designs shown in this applications note are typical requirements of any microprocessor system. The 5C060 provided a single chip solution to bind together the primary elements of that system. Few other types of programmable logic could implement the same logic in a single package. None could do it in CMOS erasable logic. The 5C060 has room for more.

APPENDIX

ADF-1

```
JR Donnell
Intel
January 31, 1986
5C060
0
5C060
Decoder for 80C88 system - 16K RAM and upper 512K EPROM
LR Version 3.0, Rescling 17x, 9/26/85
PART: 5C060
INPUTS: A19,A13,A12,PWRDWN,S2MIO,AMWC,MRDC
OUTPUTS: RAM0CS,RAM4KCS,RAM8KCS,RAM16KCS,RPROMCS
NETWORK:
RAM0CS = CONF (RAM0CS,VCC)
RAM4KCS = CONF (RAM4KCS,VCC)
RAM8KCS = CONF (RAM8KCS,VCC)
RAM16KCS = CONF (RAM16KCS,VCC)
RPROMCS = CONF (EPROMCS,VCC)
A19 = INP (A19)
A13 = INP (A13)
A12 = INP (A12)
PWRDWN = INP (PWRDWN)
S2MIO = INP (S2MIO)
MRDC = INP (MRDC)
AMWC = INP (AMWC)
EQUATIONS:
RAM8KCS = /(/MRDC*Y2
        + /AMWC*Y2):
RAM16KCS = /(/MRDC*Y3
         + /AMWC*Y3):
RPROMCS = /(/Y7
        + /Y6
        + /Y5
        + /Y4):
Y7 = /(A19*A13*A12*RNABLE):
Y6 = /(A19*A13*/A12*ENABLE):
Y5 = /(A19*/A13*A12*RNABLE):
Y4 = /(A19*/A13*/A12*ENABLE):
RNABLE = /PWRDWN*S2MIO:
Y3 = /(/A19*A13*A12*ENABLE):
Y2 = /(/A19*A13*/A12*RNABLE):
RAM4KCS = /(/MRDC*Y1
        + /AMWC*Y1):
Y1 = /(/A19*/A13*A12*ENABLE):
RAM0CS = /(/MRDC*Y0
       + /AMWC*Y0):
Y0 = /(/A19*/A13*/A12*RNABLE):
END$
```

292009-5

ADF-2

```
JR Donnell
Intel
January 31, 1986
5C060
0
5C060
Decoder for 80C88 system - 16K RAM and upper 512K EPROM
Plus power down circuit
LB Version 3.0, Baseline 17x, 9/26/85
PART: 5C060
INPUTS: A19,A13,A12,PWRDWN,S2MIO,AMWC,MRDC,82C84CLK
OUTPUTS: RAM0CS,RAM4KCS,RAM8KCS,RAM16KCS,EPROMCS,STOPCLK,80C88CLK
NETWORK:
RAM0CS = CONF (RAM0CS,VCC)
RAM4KCS = CONF (RAM4KCS,VCC)
RAM8KCS = CONF (RAM8KCS,VCC)
RAM16KCS = CONF (RAM16KCS,VCC)
EPROMCS = CONF (EPROMCS,VCC)
STOPCLK,STOPCLKF = RORF (PWRDWN,82C84CLKB,GND,GND,VCC)
80C88CLK = CONF (80C88CLK,VCC)
PWRDWN = INP (PWRDWN)
82C84CLKB = CLKB (82C84CLK)
80C88CLK = OR (STOPCLKF,82C84CLK)
82C84CLK = INP (82C84CLK)
A19 = INP (A19)
A13 = INP (A13)
A12 = INP (A12)
S2MIO = INP (S2MIO)
MRDC = INP (MRDC)
AMWC = INP (AMWC)
EQUATIONS:
RAM0CS = /(/MRDC*Y0
         + /AMWC*Y0);
RAM4KCS = /(/MRDC*Y1
         + /AMWC*Y1);
RAM8KCS = /(/MRDC*Y2
         + /AMWC*Y2);
RAM16KCS = /(/MRDC*Y3
         + /AMWC*Y3);
EPROMCS = /(/Y7
         + /Y6
         + /Y5
         + /Y4);
Y0 = /(/A19*/A13*/A12*ENABLE);
Y1 = /(/A19*/A13*A12*ENABLE);
Y2 = /(/A19*A13*/A12*ENABLE);
Y3 = /(/A19*A13*A12*ENABLE);
Y7 = /(A19*A13*A12*ENABLE);
Y6 = /(A19*A13*/A12*ENABLE);
Y5 = /(A19*/A13*A12*ENABLE);
Y4 = /(A19*/A13*/A12*ENABLE);
ENABLE = /PWRDWN*S2MIO;
END$
```

292009-6

ADF-3

```
JR Donnell
Intel
January 31, 1986
5C060
0
5C060
Decoder for 80C88 system - 16K RAM and upper 512K EPROM
Plus power down circuit
Plus wait state circuit
LR Version 3.0, Baseline 17x, 9/26/85
PART: 5C060
INPUTS: A19,A13,A12,PWRDWN,S2MIO,AMWC,MRDC,82C84CLK,ALE,WAITCS
OUTPUTS: RAM0CS,RAM4KCS,RAM8KCS,RAM16KCS,EPROMCS,STOPCLK,80C88CLK,82C84RDY
NETWORK:
RAM0CS = CONF (RAM0CS,VCC)
RAM4KCS = CONF (RAM4KCS,VCC)
RAM8KCS = CONF (RAM8KCS,VCC)
RAM16KCS = CONF (RAM16KCS,VCC)
EPROMCS = CONF (EPROMCS,VCC)
STOPCLK,STOPCLKF = RORF (PWRDWN,82C84CLKB,GND,GND,VCC)
80C88CLK,80C88CLKF = COIF (80C88CLK,VCC)
82C84RDY = RONF (82C84RDYD,80C88CLKB,82C84RDYC,GND,VCC)
PWRDWN = INP (PWRDWN)
82C84CLKB = CLKB (82C84CLK)
80C88CLK = OR (STOPCLKF,82C84CLK)
82C84CLK = INP (82C84CLK)
A19 = INP (A19)
A13 = INP (A13)
A12 = INP (A12)
S2MIO = INP (S2MIO)
MRDC = INP (MRDC)
AMWC = INP (AMWC)
80C88CLKB = CLKB (80C88CLKF)
WAITCS = INP (WAITCS)
ALE = INP (ALE)
EQUATIONS:
RAM0CS = /(/MRDC*Y0
         + /AMWC*Y0);
RAM4KCS = /(/MRDC*Y1
         + /AMWC*Y1);
RAM8KCS = /(/MRDC*Y2
         + /AMWC*Y2);
RAM16KCS = /(/MRDC*Y3
         + /AMWC*Y3);
EPROMCS = /(/Y7
         + /Y6
         + /Y5
         + /Y4);
Y0 = /(/A19*/A13*/A12*ENABLE);
Y1 = /(/A19*/A13*A12*ENABLE);
Y2 = /(/A19*A13*/A12*ENABLE);
Y3 = /(/A19*A13*A12*ENABLE);
Y7 = /(A19*A13*A12*ENABLE);
Y6 = /(A19*A13*/A12*ENABLE);
Y5 = /(A19*/A13*A12*ENABLE);
Y4 = /(A19*/A13*/A12*ENABLE);
ENABLE = /PWRDWN*S2MIO;
82C84RDYD = /82C84RDYC;
82C84RDYC = /WAITCS*ALE;
END$
```

292009-7

LEF-3

```
JR Donnell
Intel
January 31, 1986
5C060
0
5C060
Decoder for 80C88 system - 16K RAM and upper 512K EPROM
Plus power down circuit
Plus wait state circuit
LB Version 3.0, Baseline 17x, 9/26/85
PART:
        5C060
INPUTS:
        A19, A13, A12, PWRDWN, S2MIO, AMWC, MRDC, 82C84CLK, ALE, WAITCS
OUTPUTS:
        RAM0CS, RAM4KCS, RAM8KCS, RAM16KCS, EPROMCS, STOPCLK, 80C88CLK,
        82C84RDY
NETWORK:
        A19 = INP(A19)
        A13 = INP(A13)
        A12 = INP(A12)
        PWRDWN = INP(PWRDWN)
        S2MIO = INP(S2MIO)
        AMWC = INP(AMWC)
        MRDC = INP(MRDC)
        82C84CLK = INP(82C84CLK)
        ALE = INP(ALE)
        WAITCS = INP(WAITCS)
        RAM0CS = CONF(RAM0CS, VCC)
        RAM4KCS = CONF(RAM4KCS, VCC)
        RAM8KCS = CONF(RAM8KCS, VCC)
        RAM16KCS = CONF(RAM16KCS, VCC)
        EPROMCS = CONF(EPROMCS, VCC)
        ..SG000D = CLKB(82C84CLKB)
        STOPCLK, STOPCLKF = RORF(PWRDWN, ..SG000D, GND, GND, VCC)
        80C88CLK, 80C88CLKF = COTF(80C88CLK, VCC)
        ..SG001D = CLKB(80C88CLKB)
        82C84RDY = RONF(82C84RDYD, ..SG001D, 82C84RDYC, GND, VCC)
EQUATIONS:
        82C84RDYC = WAITCS' * ALE;

        ..SG001D = 80C88CLKF;

        82C84RDYD = (WAITCS' * ALE)';

        80C88CLK = (STOPCLKF' * 82C84CLK')';

        ..SG000D = 82C84CLK;

        EPROMCS = (A19 * PWRDWN' * S2MIO)';

        RAM16KCS = MRDC * AMWC
                 + A19' * A13 * A12 * PWRDWN' * S2MIO;

        RAM8KCS = MRDC * AMWC
                + A19' * A13 * A12' * PWRDWN' * S2MIO;

        RAM4KCS = MRDC * AMWC
                + A19' * A13' * A12 * PWRDWN' * S2MIO;

        RAM0CS = MRDC * AMWC
               + A19' * A13' * A12' * PWRDWN' * S2MIO;

END$
```

292009-8

RPT-3

```
Logic Optimizing Compiler Utilization Report

 ***** Design implemented successfully

JR Donnell
Intel
January 31, 1986
5C060
0
5C060
Decoder for 80C88 system - 16K RAM and upper 512K EPROM
Plus power down circuit
Plus wait state circuit
LB Version 3.0. Baseline 17x. 9/26/85

              5C060
            - - - - -
       GND -| 1   24|- Vcc
    PWRDWN -| 2   23|- A19
       GND -| 3   22|- STOPCLK
       GND -| 4   21|- R2CR4RDY
    WAITCS -| 5   20|- 80C88CLK
       ALE -| 6   19|- RPROMCS
  R2C84CLK -| 7   18|- RAM16KCS
      MRDC -| 8   17|- RAM8KCS
      AMWC -| 9   16|- RAM4KCS
     S2MIO -|10   15|- RAM0CS
       A12 -|11   14|- A13
       GND -|12   13|- GND
            -•- - - -
**INPUTS**
```

Name	Pin	Resource	MCell #	PTerms	MCells	Feeds: OR	Clear	Clock
PWRDWN	2	INP	-	-	1 4 5 6 7 8	-	-	-
WAITCS	5	INP	11	0/ 8	2	-	-2	-
ALE	6	INP	12	0/ 8	2	-	2	-
R2CR4CLK	7	INP	13	0/ 8	3	-	-	1
MRDC	8	INP	14	0/ 8	5 6 7 8	-	-	-
AMWC	9	INP	15	0/ 8	5 6 7 8	-	-	-
S2MIO	10	INP	16	0/ 8	4 5	-	-	-

292009-9

```
                                                                  6
                                                                  7
                                                                  8

        A12    11        INP          -          -               5        -        -        -
                                                                  6
                                                                  7
                                                                  8

        A13    14        INP          -          -               5        -        -        -
                                                                  6
                                                                  7
                                                                  8

        A19    23        INP          -          -               4        -        -        -
                                                                  5
                                                                  6
                                                                  7
                                                                  8

**OUTPUTS**

                                                          :                Feeds:
        Name   Pin   Resource    MCell #    PTerms       : MCells      OR    Clear   Clock

      RAM0CS    15        CONF         8       2/ 8      :    -        -        -        -
    RAM4KCS     16        CONF         7       2/ 8      :    -        -        -        -
    RAM8KCS     17        CONF         6       2/ 8      :    -        -        -        -
   RAM16KCS     18        CONF         5       2/ 8      :    -        -        -        -
    RPROMCS     19        CONF         4       1/ 8      :    -        -        -        -
   80CRRCLK     20        COTF         3       1/ 8      :    -        -        -        2
   82C84RDY     21        RONFA        2       1/ 8      :    -        -        -        -
    STOPCLK     22        RORFA        1       1/ 8      :    3        -        -        -

**UNUSED RESOURCES**

        Name   Pin   Resource    MCell      PTerms

          -      1          -        -           -
          -      3          -        9           8
          -      4          -       10           8
          -     13          -        -           -

**PART UTILIZATION**

  81%      Pins
  87%      MacroCells
  9%       Pterms
```

292009-10

APPLICATION NOTE

AP-276

June 1986

Implementing a CMOS Bus Arbiter/Controller in the 5C060 EPLD

DANIEL E. SMITH
APPLICATIONS ENGINEERING
INTEL CORPORATION

Order Number: 292012-001

INTRODUCTION

This application note shows how to implement a CMOS Bus Arbiter/Controller in an Intel 5C060 EPLD (Erasable Programmable Logic Device). The note includes a brief overview of a similar circuit implemented with typical PLA devices, a more detailed discussion of the 5C060 implementation, and a summary.

The bus priority resolution and arbitration scheme selected for the circuit is that used by the industry-standard MULTIBUS I interface. Operation and timing for the MULTIBUS I interface is well understood by most engineers and is described in readily available Intel publications. Thus, a description of the MULTIBUS I interface is not included here. The bus arbiter/controller functions shown here support both serial and parallel priority resolution between bus masters. Timing is equivalent to MULTIBUS I specifications. Electrical specifications for both the PLA and EPLD approaches vary from MULTIBUS I standards. Neither of the two circuits discussed here provide the full current sink capability for **all** MULTIBUS I signals. Because the EPLD implementation is designed for CMOS systems, however, this requirement is not relevant for the 5C060 implementation.

PLA APPROACH

The functional equivalent of a MULTIBUS I arbiter/controller can be implemented in two 20-pin PLA-type devices as shown in Figures 1 and 2. (Figure 1 shows the logic for the arbiter device. Figure 2 shows the logic for the controller and the connections to the arbiter.) Figure 3 shows the arbiter list file as an example of PLA-type files. Two different 20-pin PLA devices are required to implement the arbiter and controller functions, a 16R4-type device and a 16L8-type device.

Implementation of logic devices in PLA-type devices, such as those shown here, has proven to be quite beneficial. Development time and cost is much less than for custom silicon device designs. The two PLA-type devices take up less board space than a discrete TTL implementation of the same functions. In addition, the two raw devices can also be used for different functions in other products, thereby reducing inventory costs. As a result of these factors (and others), use of PLA-type devices has grown substantially in recent years.

With the increased density and flexibility of EPLD devices over typical PLA-type devices, even greater space, inventory, and cost savings can be obtained by using EPLDs. The following section shows an implementation of the same arbiter/controller functions in a single 24-pin 5C060 EPLD device.

5C060 IMPLEMENTATION

The equivalent functions for both the MULTIBUS I arbiter and controller fit inside a **single** 5C060 EPLD device. The 5C060 device is available in a 24-pin 0.3" DIP package. Figures 4 and 5 show logic diagrams for the arbiter and controller functions. When compared with the PLA implementation, some differences in the design are immediately apparent. These differences result from the characteristics of the EPLD macrocell or from corrections to the circuit used in Figures 1 and 2.

The major change resulting from the EPLD macrocell structure concerns the EPLD output buffers. Since output buffers from macrocells are non-inverting (PLA-type devices typically contain inverting buffers), signals enter the buffers in the same logic orientation from which they are to appear at the output. The logic for the EPLD (shown in Figures 4 and 5) incorporates this change.

Some errors in the PLA-type implementation have also been corrected in the EPLD design. These changes are as follows:

- The M/$\overline{IO}$ input to the MRDC/ and MWTC/ gates is inverted. M/$\overline{IO}$ distinguishes between memory and I/O cycles. The PLA-type implementation does not use this signal properly; the PLA-type controller generates read or write commands to both memory and I/O at the same time, which can result in contention between memory and I/O during bus transfers.
- BPRO/ is gated by BPRN/ in the EPLD design. When using serial priority resolution, this allows the highest priority arbiter to prevent all other masters from controlling the bus. (In the PLA design, BPRO/ is enabled/disabled only by a local request. Higher priority arbiters cannot disable all other arbiters. This can result in contention between bus masters. By gating BPRO/ with BPRN/ in the EPLD design, this source of bus contention is prevented.)

Figure 6 shows the list file for the arbiter/controller device. Figure 7 shows the report file produced by the iPLDS software. This file contains a pinout diagram of the final programmed device and provides a resource usage map for the device.

Most of the input and output signals are self-explanatory to those familiar with Intel processors and the MULTIBUS I interface. The XREQ input is the bus transfer request signal from the address decode logic. The BUSY/ and CBRQ/ outputs are bi-directional, simulated open-collector outputs. These outputs use the iPLDS 5C060 (Combinational-Output I/O-Feedback) primitive in the list file. The BUSY/ signal serves to illustrate this use of EPLD outputs.

A pull-up resistor is used externally (i.e., on the backplane) to hold BUSY/ high when no arbiter is in control of the bus. When the arbiter is granted control of the bus, AEN is clocked high, which enables the output of the BUSY/ driver. Since the input to the BUSY/ driver is low during normal operation (RESET/ inverted), the enabled driver pulls BUSY/ low to signal other arbiters that the bus is in use. When the arbiter is finished using the bus, AEN goes low to disable the BUSY/ driver (three-state output). The pull-up resistor pulls BUSY/ high to signal other arbiters that the bus is free for use if needed.

Note that BUSY/ is also routed into the bus grant logic as input BSI. BSI prevents the arbiter from taking control of the bus (and driving BUSY/ low) when some other arbiter already has control of the bus. Thus only one arbiter may pull BUSY/ low at any one time.

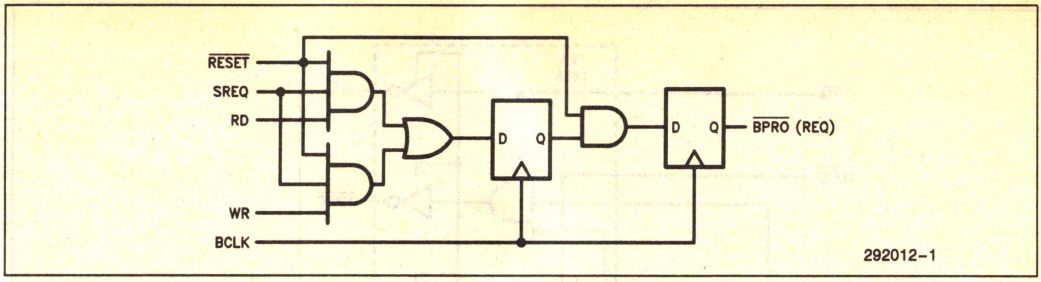
The one difference between standard MULTIBUS I logic levels and the EPLD implementation described here relates to the BCLK/ signal. MULTIBUS I bus arbitration uses the negative-going edge of BCLK/ to synchronize events. All 5C060 flip-flops, however, clock on the positive-going edge of BCLK/. If all bus masters in the system use the same arbiter implementation, this poses no problem. Otherwise, an external inverter is required for the BCLK/ input.

COMPARISON/SUMMARY

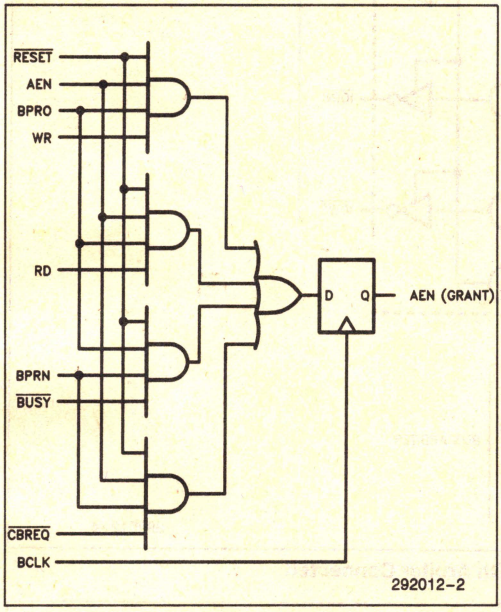
Both the PLA and EPLD implementations of the bus arbiter/controller result in a lower device count than a discrete logic circuit. Lower device count means less p.c. board space, fewer assembly steps, and fewer device interconnects. Both PLA and EPLD implementations are quicker and less expensive to develop than a custom gate array or dedicated silicon device.

In contrast to the PLA approach, however, the EPLD implementation requires only a single device, while the PLA approach requires two different devices. Thus the EPLD approach results in twice the cost savings (inventory and assembly) and half the programming activity to produce the device. Fewer device interconnects also means greater reliability. In addition, programmed EPLD devices can be erased and reprogrammed for a different application if needed, a feature not available with PLAs.

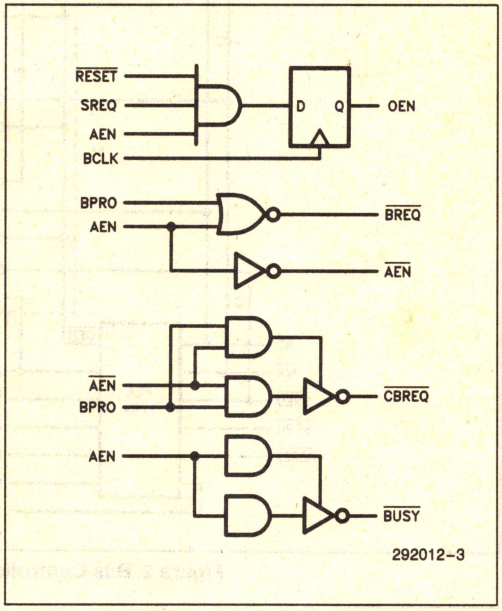
Overall, the greater flexibility, and the incremental design, manufacturing, and cost advantages of EPLD devices make them ideal for many applications where PLA devices would otherwise be used.



A) Request Synchronizer



B) Grant/Access Logic



C) Bus Transfer Control

Figure 1. PLA Approach to a Bus Arbiter

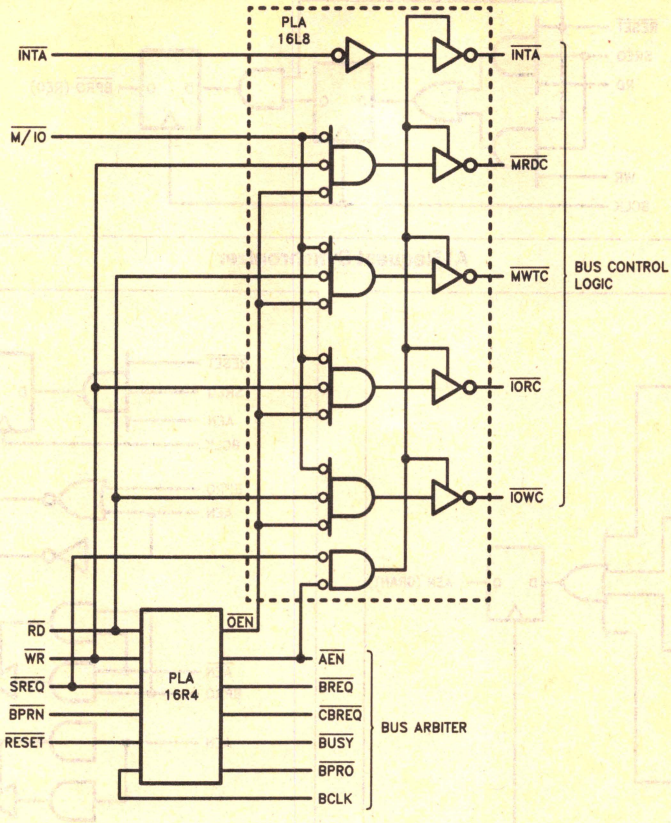
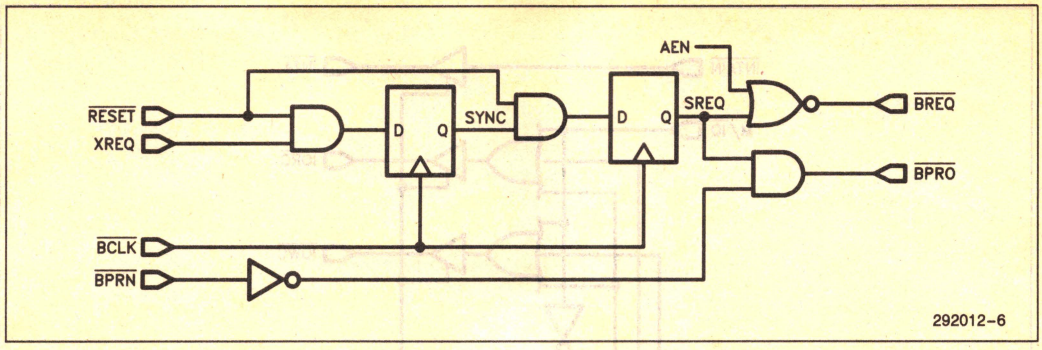


Figure 2. Bus Controller with Arbiter Connected

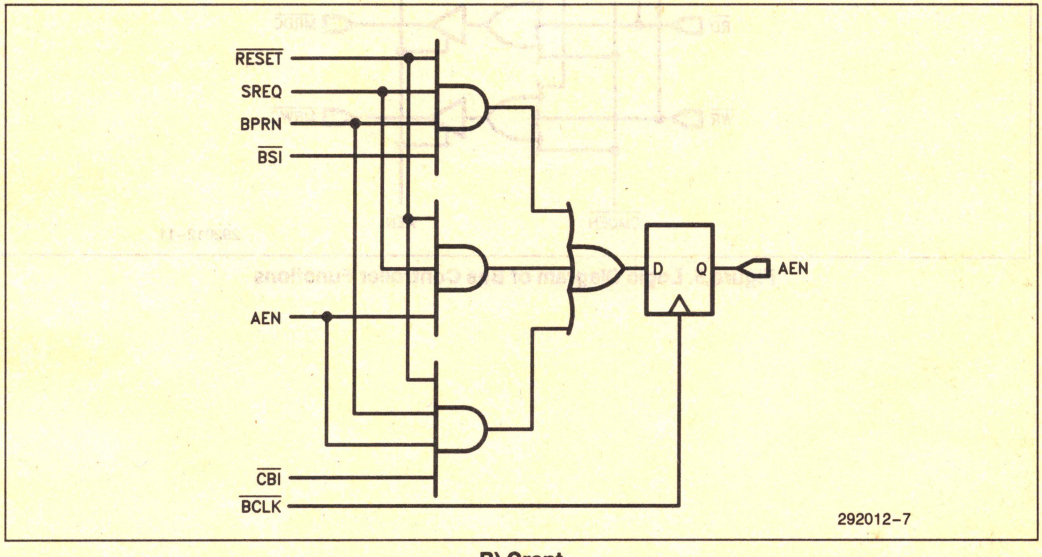
```
PLA16R4                                          PLA DESIGN FILE
ARB001                                        D. E. ENGR.  1/1/85
MULTIBUS I ARBITER
SOME SYSTEM COMPANY
BCLK  /WR    /RD   /SREQ /RESET /BPRN  NC   NC    NC  GND
/E    /CBREQ /BUSY /SYNC /BPRO  /AEN   /OEN /BREQ NC  VCC

SYNC  := /RESET*SREQ*WR          +
         /RESET*SREQ*RD

BPRO  := /RESET*SYNC

AEN   := /RESET* AEN*BPRO*WR     +
         /RESET* AEN*BPRO*RD     +
         /RESET*BPRO*BPRN*/BUSY  +
         /RESET* AEN*BPRN*/CBREQ

OEN   := /RESET*SREQ*AEN

IF(BPRO*/AEN) CBREQ = BPRO*/AEN

IF(AEN) BUSY = AEN

BREQ   = BPRO +
         AEN
```

292012-5

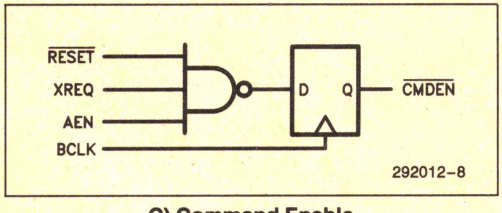
Figure 3. List File for PLA Arbiter



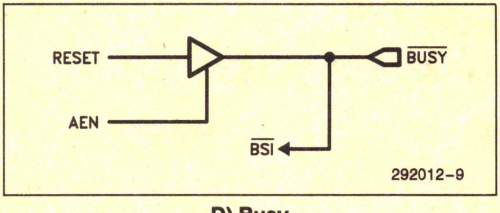
A) Request



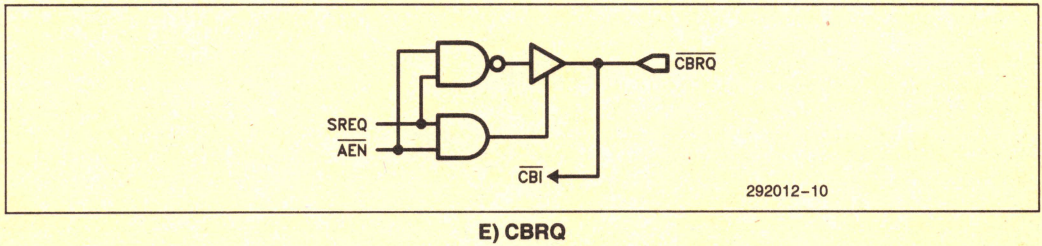
B) Grant



C) Command Enable



D) Busy



E) CBRQ

Figure 4. Logic Diagram of Bus Arbiter Functions

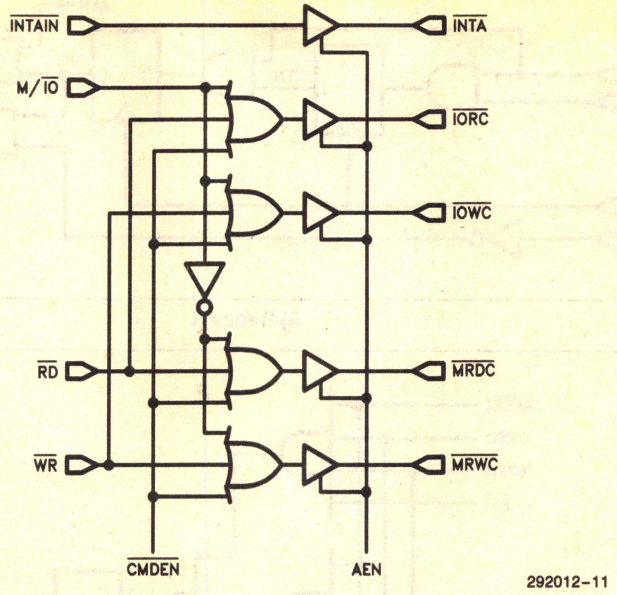


Figure 5. Logic Diagram of Bus Controller Functions

```
DANIEL E. SMITH
INTEL CORPORATION
MARCH 27, 1986
VERSION 1.1
REV. A
5C060
CMOS BUS ARBITER/CONTROLLER


PART:     5C060
INPUTS:   BCLK, XREQ, RESET, BPRN, MIO,  RD,   WR,   INTAIN
OUTPUTS:  BPRO, AEN,  BREQ,  CBRQ, BUSY, INTA, MRDC, MWTC, IORC, IOWC


NETWORK:

BCLK      = INP (BCLK)                    %BUS CLOCK INPUT%
INTAIN    = INP (INTAIN)                  %INT. ACK. INPUT%
XREQ      = INP (XREQ)                    %SYSTEM REQUEST INPUT%
RESET     = INP (RESET)                   %RESET INPUT%
BPRN      = INP (BPRN)                    %BUS PRIORITY INPUT%
MIO       = INP (MIO)                     %MEMORY/IO INPUT%
RD        = INP (RD)                      %READ INPUT%
WR        = INP (WR)                      %WRITE INPUT%
BPRO      = CONF (BPROc,VCC)              %BUS PRIORITY OUTPUT%
AEN,AEN   = RORF (AENd,BCLK,GND,GND,VCC)  %ADDRESS ENABLE (GRANT)%
BREQ      = CONF (BREQc,VCC)              %BUS REQUEST%
CBRQ,CBI  = COIF (CBRQc1,CBRQc2)          %CBRQ/ -- SIMULATED O.C.%
BUSY,BSI  = COIF (BUSYc,AEN)              %BUSY/ -- SIMULATED O.C.%
INTA      = CONF (INTAIN,AEN)             %INT. ACK. OUTPUT%
MRDC      = CONF (MRDCc,AEN)              %MEMORY READ COMMAND%
MWTC      = CONF (MWTCc,AEN)              %MEMORY WRITE COMMAND%
IORC      = CONF (IORCc,AEN)              %I/O READ COMMAND%
IOWC      = CONF (IOWCc,AEN)              %I/O WRITE COMMAND%
SREQ      = NORF (SREQd,BCLK,GND,GND)     %VALID BUS REQUEST%
SYNC      = NORF (SYNCd,BCLK,GND,GND)     %SYNCHRONIZED REQUEST%
CMDEN     = NORF (CMDENd,BCLK,GND,GND)    %COMMAND ENABLE%
```

292012-12

```
EQUATIONS:

BPROc     = (SREQ * /BPRN);
AENd      = RESET * SREQ * /BPRN * BSI +
            RESET * SREQ * AEN +
            RESET * /BPRN * AEN * CBI;
BREQc     = /(SREQ + AEN);
BUSYc     = /RESET;
CBRQc1    = /(SREQ * /AEN);
CBRQc2    = SREQ * /AEN;
MRDCc     = /MIO + RD + CMDEN;
MWTCc     = /MIO + WR + CMDEN;
IORCc     = MIO + RD + CMDEN;
IOWCc     = MIO + WR + CMDEN;
SREQd     = RESET * SYNC;
SYNCd     = RESET * XREQ;
CMDENd    = /(RESET * XREQ * AEN);


END$
```

292012-13

Figure 6. iPLDS Network List File

```
Logic Optimizing Compiler Utilization Report

 ***** Design implemented successfully

DANIEL E. SMITH
INTEL CORPORATION
MARCH 27, 1986
VERSION 1.1
REV. A
5C060
CMOS BUS ARBITER/CONTROLLER


                5C060
              _ _ _ _ _
        BCLK -| 1   24|- Vcc
         MIO -| 2   23|- XREQ
  RESERVED -| 3   22|- INTA
  RESERVED -| 4   21|- IOWC
  RESERVED -| 5   20|- IORC
         AEN -| 6   19|- MWTC
        BPRO -| 7   18|- MRDC
      INTAIN -| 8   17|- BUSY
          WR -| 9   16|- CBRQ
          RD -|10   15|- BREQ
        BPRN -|11   14|- RESET
         GND -|12   13|- GND
              - - - - -


**INPUTS**

                                                                  |          Feeds:
      Name  Pin  Resource   MCell #    PTerms   |  MCells      OE   Clear   Clock

       BCLK    1       INP         -         -   |       -       -       -    CLK1

        MIO    2       INP         -         -   |       2       -       -       -
                                                  |       3
                                                  |       4
                                                  |       5

     INTAIN    8       INP        14      0/ 8   |       1       -       -       -

         WR    9       INP        15      0/ 8   |       2       -       -       -
                                                  |       4

         RD   10       INP        16      0/ 8   |       3       -       -       -
                                                  |       5

       BPRN   11       INP         -         -   |      12       -       -       -
                                                  |      13

      RESET   14       INP         -         -   |       6       -       -       -
                                                  |       9
                                                  |      10
                                                  |      11
                                                  |      12

       XREQ   23       INP         -         -   |       9       -       -       -
                                                  |      10
```

292012-14

Figure 7. iPLDS Report File

```
**OUTPUTS**

                                                          Feeds:
Name   Pin   Resource   MCell #   PTerms   :   MCells      OE    Clear   Clock
AEN      6       RORF        12      3/ 8       7         -7      -       -
                                                8          1
                                                9          2
                                               12          3
                                                           4
                                                           5
                                                           6
BPRO     7       CONF        13      1/ 8       -          -       -       -
BREQ    15       CONF         8      1/ 8       -          -       -       -
CBRQ    16       COIF         7      1/ 8      12          -       -       -
BUSY    17       COIF         6      1/ 8      12          -       -       -
MRDC    18       CONF         5      1/ 8       -          -       -       -
MWTC    19       CONF         4      1/ 8       -          -       -       -
IORC    20       CONF         3      1/ 8       -          -       -       -
IOWC    21       CONF         2      1/ 8       -          -       -       -
INTA    22       CONF         1      1/ 8       -          -       -       -

**BURIED REGISTERS**

                                                          Feeds:
Name   Pin   Resource   MCell #   PTerms   :   MCells      OE    Clear   Clock
         3       NORF         9      1/ 8       2         -       -       -
                                                3
                                                4
                                                5
         4       NORF        10      1/ 8      11         -       -       -
         5       NORF        11      1/ 8       7         7       -       -
                                                8
                                               12
                                               13

**UNUSED RESOURCES**

     Name   Pin   Resource     MCell     PTerms

       -     13          -         -          -

**PART UTILIZATION**

95%      Pins
100%     MacroCells
11%      Pterms
```

292012-15

Figure 7. iPLDS Report File (Continued)

APPLICATION NOTE

AP-304

March 1987

Simulation of EPLD Timing

PEDRO VARGAS
PROGRAMMABLE LOGIC APPLICATIONS
INTEL CORPORATION

Order Number: 292027-001

INTRODUCTION

Though there are many important activities that are considered in a design, timing analysis usually heads the list when it comes to evaluating functionality and performance. Timing issues are prevalent during design, and at reviews when worst case analysis is performed. By being familiar with timing specifics of EPLD architecture, the designer can assess timing issues throughout the design phase.

OBJECTIVE

This application note details the internal timing of Intel EPLDs. It breaks down the internal architecture into functional timing elements to extract timing data, and then presents a method of timing simulation. The relationship of these elements to data sheet parameters is also shown by several examples. By applying these concepts, engineers will be able to simulate their designs and have a better understanding of EPLD timing.

EPLD STRUCTURE

Intel EPLDs consist of a programmable logic array and a configurable I/O block as shown in Figure 1. The array is composed of two-level logic, incorporating a programmable AND array and a fixed OR array. The AND matrix is a crosshatch of the true and complements of all the pin inputs and the AND array inputs. At each intersection there exists an EPROM cell that determines if that input feeds the AND gate. By selectively programming these EPROM cells, complex logic functions are implemented in the familiar sum of products form. The output of the OR gate feeds an I/O architecture block that has a variety of programmable options. Combined, the logic array and I/O block is called a macrocell. Each macrocell output exits via an I/O pin.

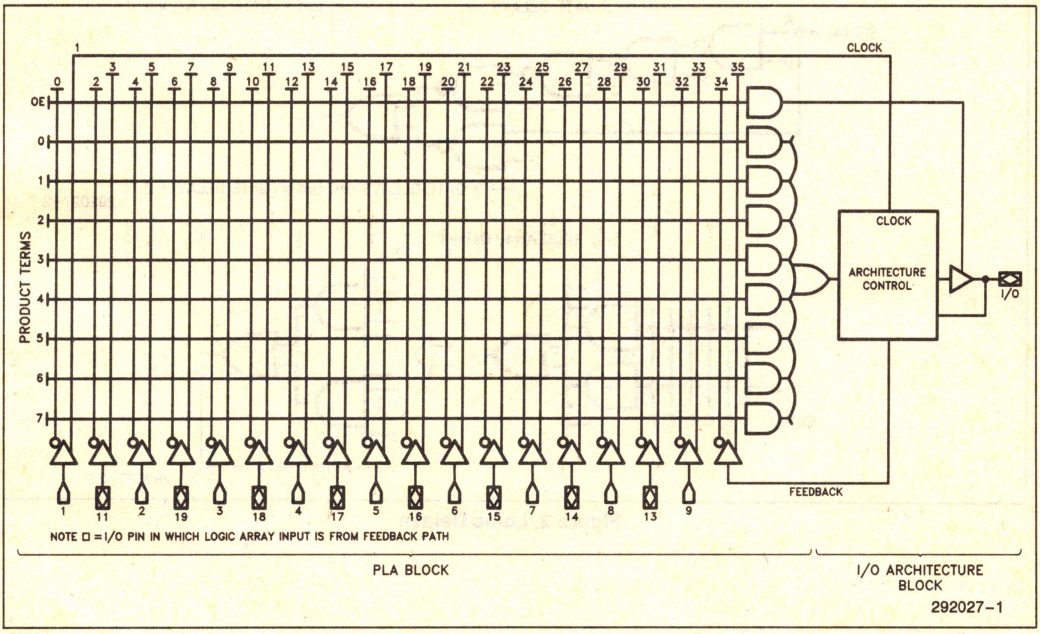


Figure 1. EPLD Macrocell

EPLDs have two specifications that influence delays within the component, maximum propagation delay (tPD), and minimum clock period (tP1). Propagation delay is the time that it takes a signal to appear at the output relative to the input. tPD is defined for combinatorial outputs. Minimum clock period is the smallest allowable clock cycle that determines maximum operating frequency. Maximum operating frequency (fMAX) is defined for registered functions.

Unlike gate arrays that deal with individual gate delays, EPLDs have internal delays that are grouped differently. With a gate array, a logic function may have different speed paths for each product term, depending on the number of two input NAND gates in each path. In an EPLD, each product term is the equivalent of a multi-input AND gate. Figure 2 shows a comparison of gate delays and array delays.

The top figure shows that SIGNAL A has two cumulative speed paths in the gate array circuit. In the EPLD inplementation, each product term has the same delay and there is only one array delay. Intel EPLDs have inputs that range from 18 (5C031) to 64 (5C180). Because the parts are characterized at worst case, the array delay is the same regardless of inputs used. In the case of product terms, the EPLD family supports from 74 to 480. Here again, the number of product terms does not affect the array delay.

The I/O block varies in complexity within the EPLD family, but a typical arrangement is shown in Figure 3. I/O programmability is accomplished by configuring for register types and choosing one of several outputs or feedback paths. Timing paths and delays depend on the way the output and feedback muxes are configured.

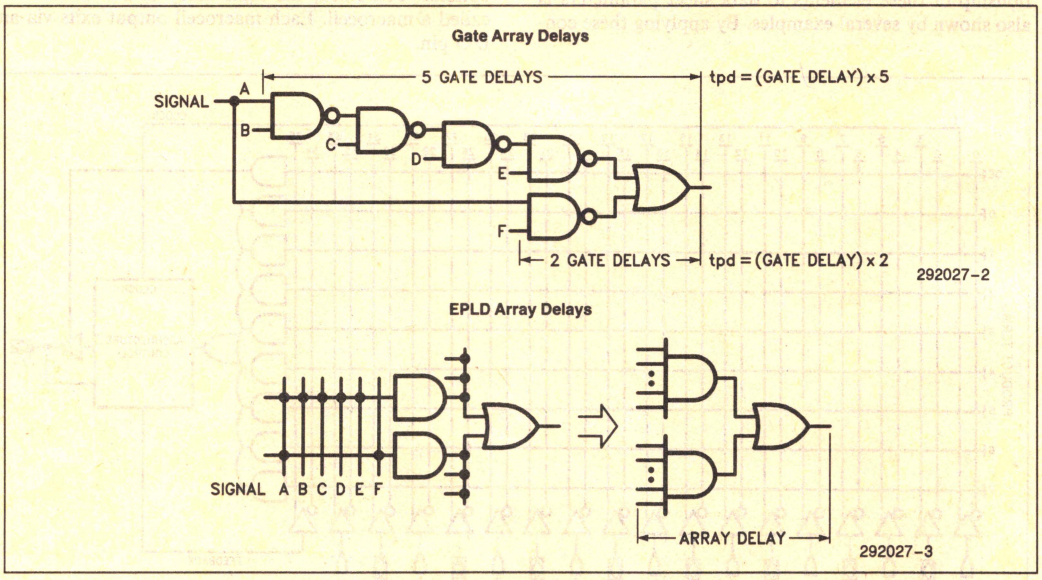
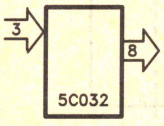


Figure 2. Logic Delays

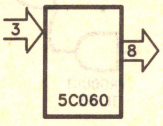
MORE ON ARRAY DELAYS

The number of inputs and product terms used in an EPLD array doesn't change the array delay because the EPROM cells are always connected, whether they are programmed or not. As a result, when a device is tested the array delay is at worst case load already. When the same design is implemented in a different EPLD, the array delay will be different, due to the different IC geometries. For example, a simple three to eight decoder implemented in a 5C032 will have an array delay of 17 ns. The same decoder when implemented in a 5C060 will have an array delay of 30 ns.



292027-4

t_{AD} = 17 ns



292027-5

t_{AD} = 30 ns

Designing an EPLD circuit involves working with the Intel iPLS[3] (Intel Programmable Logic Software) logic primitives. Logic primitives are functional building blocks that the EPLD software requires to implement a circuit. The primitives consist of input, logic, and output functions such as INP, AND, OR, CONF, RORF etc. Because of the modular nature of the design primitives, the resulting logic implementation is very modularized and lends itself well to an analysis of timing paths. The following sections detail the delays involved with each primitive.

TIMING ELEMENTS

Each EPLD macrocell can be functionally modeled with the seven blocks shown in Figure 4.

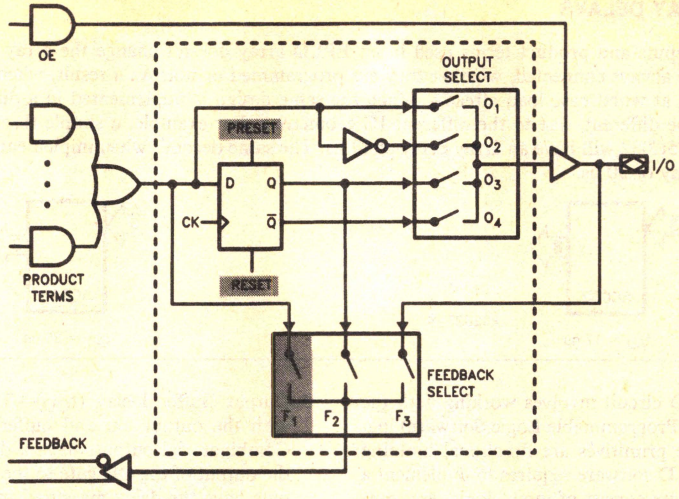
The macrocell timing model consists of input buffer delay for the inputs and the clock, array delay, register delay and output delay. The model shows the feed forward path as well as the combinatorial and registered feedback paths. The feedback paths may apply depending on the application, and whether the design is combinatorial or registered. The model also applies to devices that have global and local buses.

Combinatorial designs with no feedback contain three functional blocks for input, array, and output delays. Four blocks are required for a design with feedback. Register designs may be more complex but contain at least five blocks. These are: input, array, output, register, and clock delays. The delay for each block is defined as:

1. Input Buffer Delay (t_{IN})—The delay associated with the input pin and buffers. One delay value applies to both true and complementary buffers that drive the AND array.
2. Array Delay (t_{AD})—The time that it takes a signal to propagate through the AND array and appear at the output of the OR gate. This delay is characterized at worst case and is independent of number of inputs or product terms.
3. Output Buffer Delay (t_{OD})—The delay associated with the output pin and buffer of each macrocell. Combinatorial outputs have a delay measured from the output of the OR gate to the pin. Registered outputs have the delay measured from the register output to the pin. The delay value is the same for either output configuration.
4. Combinatorial Feedback Delay (t_{CF})—The delay from the output of the OR gate to the input of the AND array. The delay is measured when both the true and complement of the signal appear at the input of the array.
5. Register Delay (t_{RD})—The delay through any flip-flop. The delay is measured from the triggering clock edge to the time when data is valid at the output of the register.
6. Register Feedback Delay (t_{RF})—The delay from the data valid at the flip-flop output to the time it appears in true and complement form at the array input.
7. Input Clock Delay (t_{IC})—The time that the clock is delayed in reaching the input of the internal register.

Use of these delay paths depends on the EPLD output configuration. Figure 5 introduces the concept of timing elements that is used throughout this application note. Use of these elements depends on the application. If a design is combinatorial, then the only paths to consider are the input buffer, the array, the output buffer, and the combinatorial feedback path. Conversely, if a design is registered, the paths to consider are all of the previously listed delays, with the addition of the register delay, the clock delay, and the registered feedback path.

The manner in which the delay values are used is called simulation. Simulating a EPLD circuit means calculating the output timing of a device from internal timing with the aid of timing data and simulation model (like Figure 5). Before we get into simulation let's examine how these internal timing elements relate to the data sheet specifications of each device.



NOTE:
Controls shaded in gray are available on the 5C031 only.

Figure 3. I/O Architecture Control

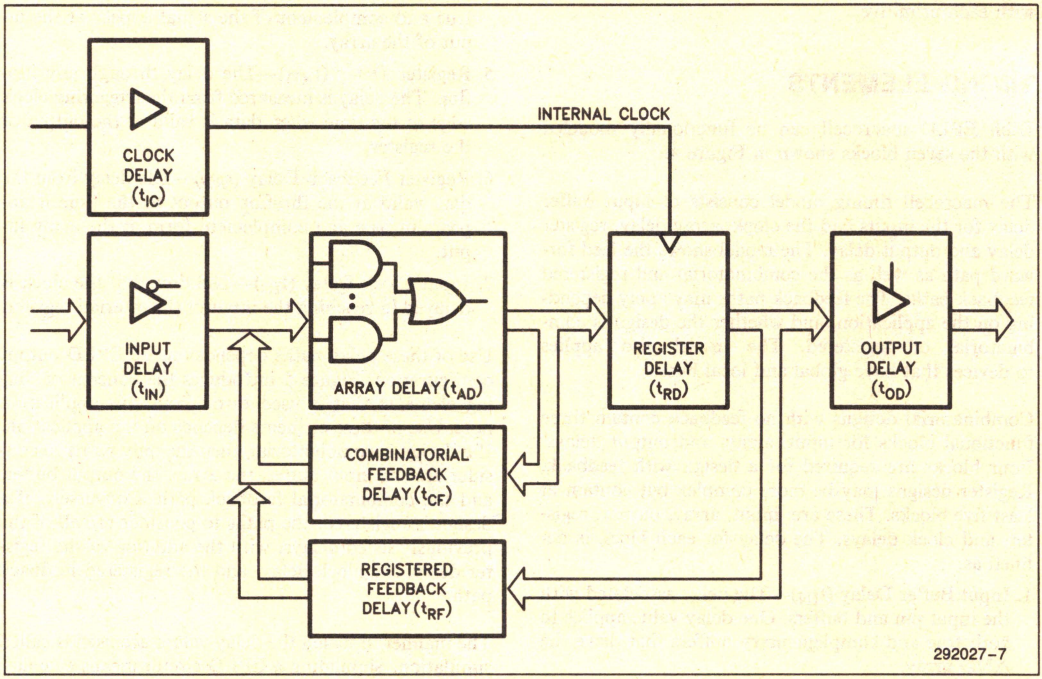


Figure 4. EPLD Delay Blocks

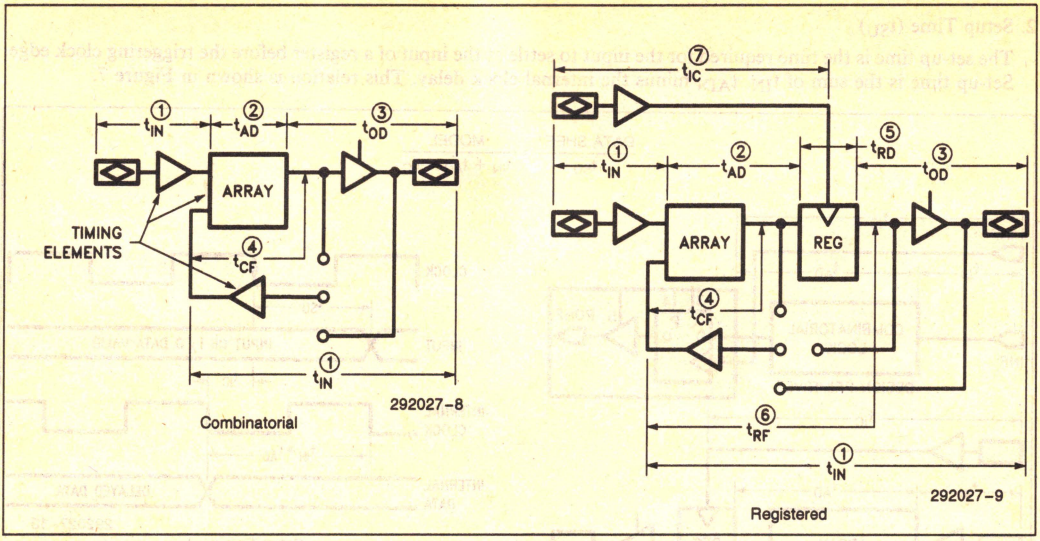


Figure 5. EPLD Delay Paths

DATA SHEET SPECIFICATIONS

Timing specifications for EPLDs are found in the data sheets under "A.C. Characteristics". Data sheet values are derived by testing a device under worst case conditions. Test conditions are both static and dynamic based on several variables like input levels, output loading, frequency, and temperature. Characterizing a device involves detailed testing of specific device functions and correlating test data to performance analysis done during design. The results are placed in the data sheets, which provide the designer with the worst case data that they may need for their application.

The timing data found in EPLD data sheets is derived from the timing elements previously described. Since an EPLD design can be broken down into either a combinatorial or registered macrocell, the data sheets contain specific information for each mode. Figures 6 through 10 correlate the timing elements to data sheet values. Each figure shows the delay path as it applies to the Intel design primitives and the simulation model.

1. Propagation Delay (t_{PD})

   Defined as the time required for an external input to travel through any combinatorial logic and appear at the external EPLD pin. This specification applies to combinatorial logic with non-registered output. Figure 6 shows that this specification is the sum of t_{IN}, t_{AD}, and t_{OD}.

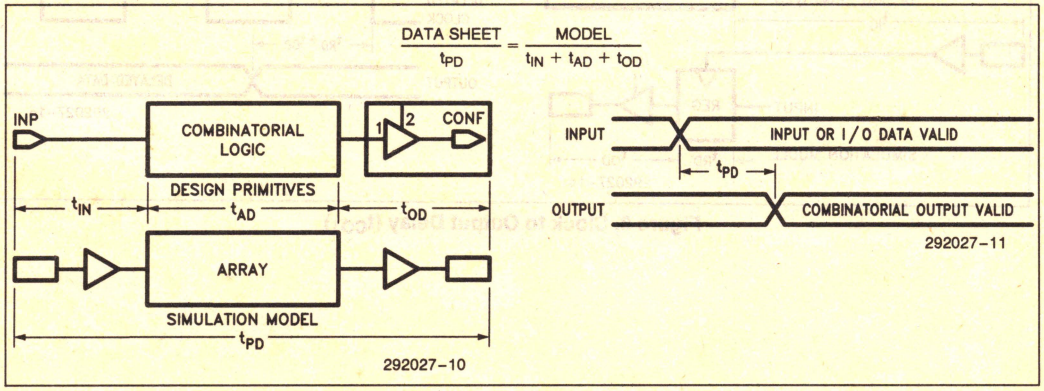


Figure 6. Propagation Delay (t_{PD})

2. Setup Time (t_{SU})

The set-up time is the time required for the input to settle at the input of a register before the triggering clock edge. Set-up time is the sum of t_{IN}, t_{AD}, minus the internal clock delay. This relation is shown in Figure 7.

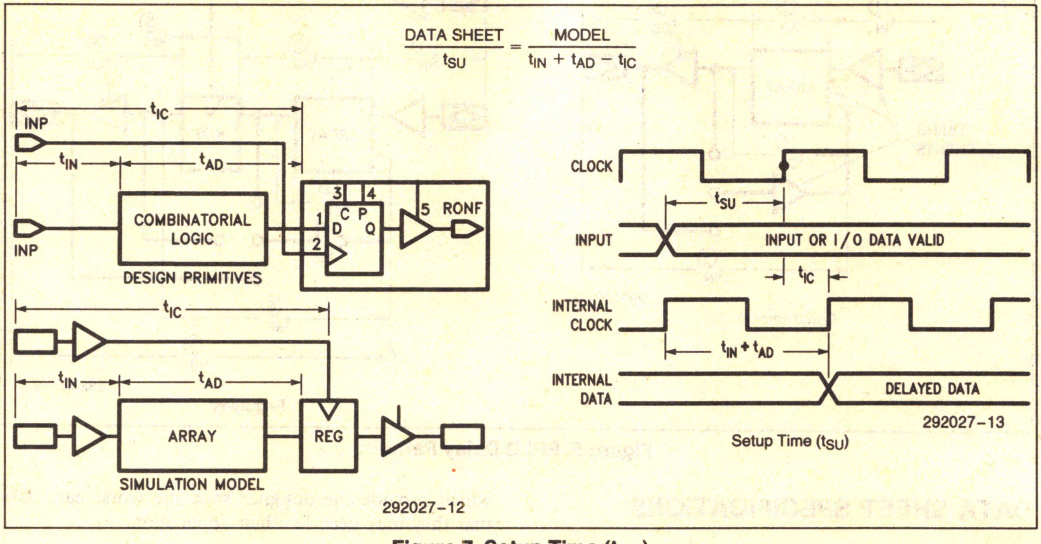


Figure 7. Setup Time (t_{SU})

3. Clock to Output Delay (t_{CO1})

Defined as the time required for a signal to pass through a register and appear at the EPLD external pin relative to the external triggering edge of the clock. This delay is the sum of t_{IC}, t_{RD}, and t_{OD} as shown in Figure 8.

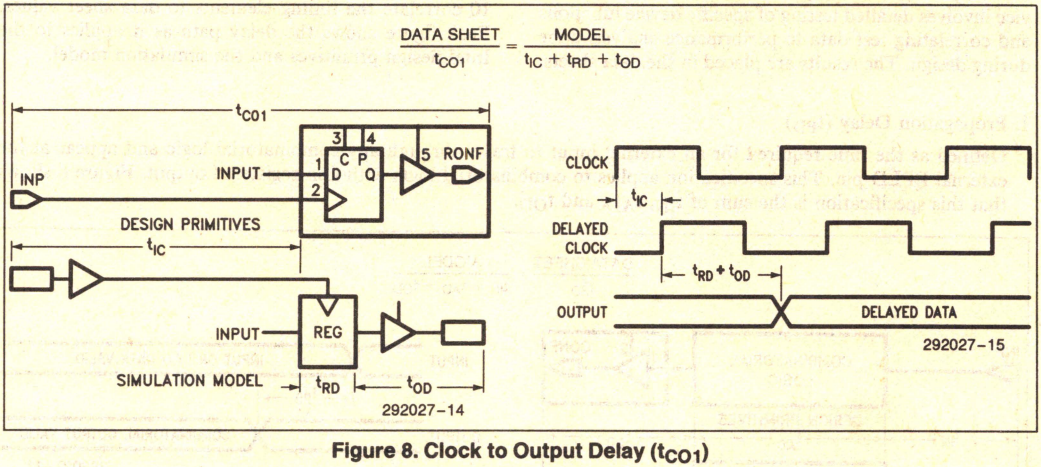


Figure 8. Clock to Output Delay (t_{CO1})

4. Minimum Internal Clock Period (t_{P1})

Defined as the maximum frequency at which an EPLD can operate when register inputs are dependent on internal logic only, and not affected by external inputs. Another way to think of this time is, as the fastest rate that a signal can be routed from register to register through the array via an internal feedback path. This minimum period is the sum of t_{RD}, t_{RF}, and t_{AD} as shown in Figure 9.

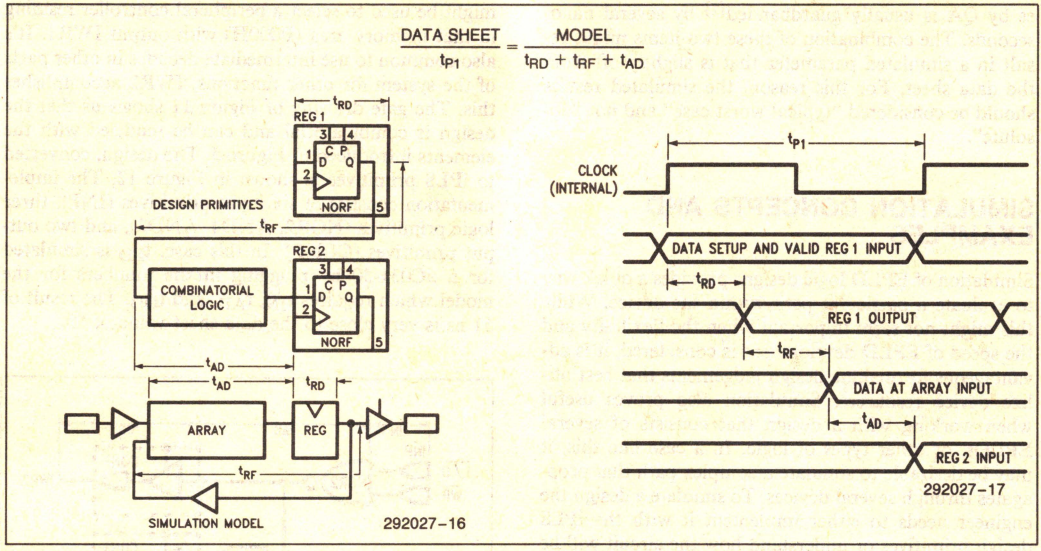


Figure 9. Minimum Internal Clock Period (t_{P1})

5. Registered Feedback to Combinatorial Output (t_{CO2})

This is the time required for an input to propagate through a register, feedback to combinatorial logic and appear at the external pin, relative to the external clock. This time is the sum of t_{IC}, t_{RD}, t_{RF}, t_{AD}, and t_{OD} as shown in Figure 10.

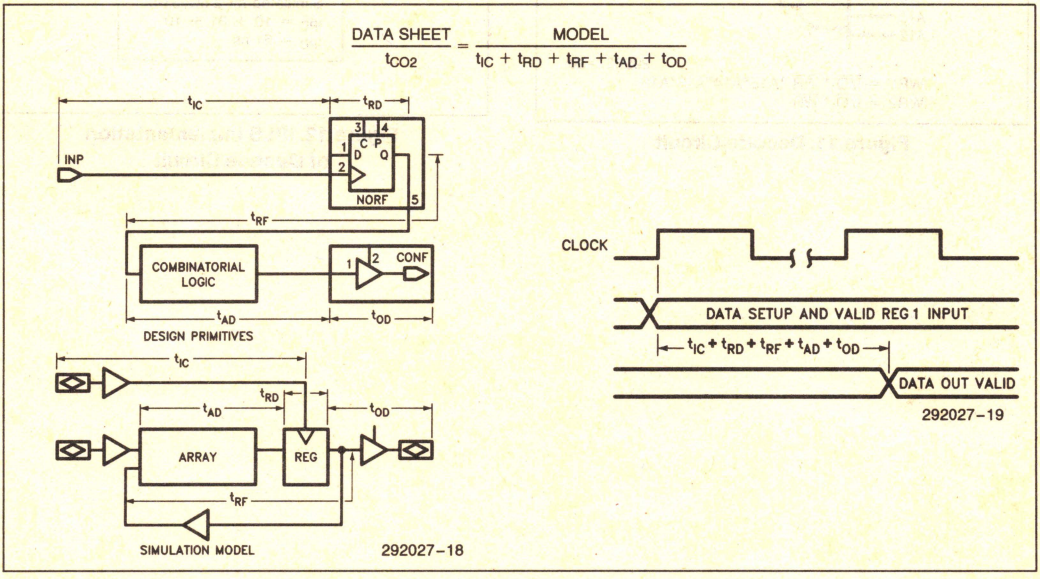


Figure 10. Registered Feedback to Combinatorial Output (t_{CO2})

When simulating EPLD designs with data from the simulation tables, it is possible that there may be a small discrepancy between simulation and data sheet values. Because our simulations deal with ideal waveforms, rise and fall times are not taken into consideration. Also, characterization and final test of these devices by QA is usually guardbanded[2] by several nanoseconds. The combination of these two items might result in a simulated parameter that is slightly off from the data sheet. For this reason, the simulated results should be considered "typical worst case" and not "absolute".

SIMULATION CONCEPTS AND EXAMPLES

Simulation of EPLD logic designs provides a quick way to evaluate a particular path within the device. While this might not seem important when the flexibility and the speed of EPLD development is considered, it is advantageous for making design judgements that best utilize device resources. Simulation also proves useful when working with a design that consists of several EPLDs, or other types of logic. In a case like this, it may be desirable to simulate a complex path that propagates through several devices. To simulate a design the engineer needs to either implement it with the iPLS design primitives or understand how the circuit will be implemented by the software. This provides the modularized layout that is needed to choose a model.

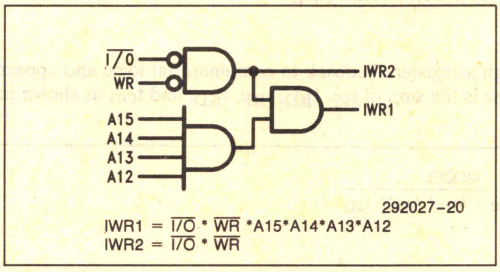


Figure 11. Decode Circuit

The decoder circuit shown in Figure 11 serves as a good starting point for simulation. Decoding two control lines (I/O*, WR*) and four addresses, the circuit might be used to select a peripheral controller residing in that memory area (F000H) with output IWR1. It's also common to use intermediate decodes in other parts of the system for other functions, IWR2 accomplishes this. The gate drawing of Figure 11 shows us that the design is combinatorial and can be modeled with the elements introduced in Figure 5. The design, converted to iPLS primitives is shown in Figure 12. The implementation consists of six input primitives (INP), three logic primitives (NOR2, AND4, AND2), and two output primitives (CONF). In this case, t_{PD} is simulated for a 5C031-50 by plugging in the numbers for the model which requires t_{IN}, t_{AD}, and t_{OD}. The result of 51 ns is very close to the data sheet value of 50.

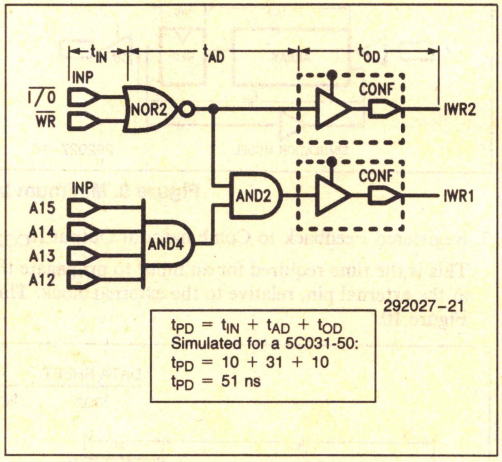


Figure 12. iPLS Implementation of Decode Circuit

The second example is the wait state circuit shown in Figure 13. The circuit shows one way that a synchronous wait state can be generated in an EPLD. This circuit would be used with a microprocessor that samples a WAIT signal on the falling edge of the T1 clock. If WAIT is valid, the micro inserts an extra cycle into the memory operation. After the wait state, the cycle ends normally. The circuit is first converted to the Intel design primitives for further observation. The iPLS design is shown in Figure 14. One difference from the previous circuit is the addition of a register primitive, in this case a NORF (No Output Registered Feedback). In this application the critical path t_{CO2} is evaluated to insure that the wait signal is sampled at the appropriate time. The modularized layout of the primitives shows that this circuit can be simulated with the registered model of Figure 5. The result is simulated for a 5C032-35 by adding t_{IC}, t_{RD}, t_{RF}, t_{AD}, and t_{OD}.

Our last example is the asynchronous R-S latch shown in Figure 15. Applications that use EPLDs without configurable output registers may use this circuit as a work around solution. The output primitive of Figure 16 is a COCF (Combinatorial Output Combinatorial Feedback) being driven by two NOR2 gates. Because combinatorial feedback to the same macrocell is being used, care must be taken that the input pulses are long enough to avoid output glitches. In this example, the input pulses should be longer than $t_{AD} + t_{CF}$ for proper latching. For this example, t_{PD} is the critical parameter. Simulation results in t_{PD} equal the sum of t_{IN}, t_{AD}, and t_{OD}.

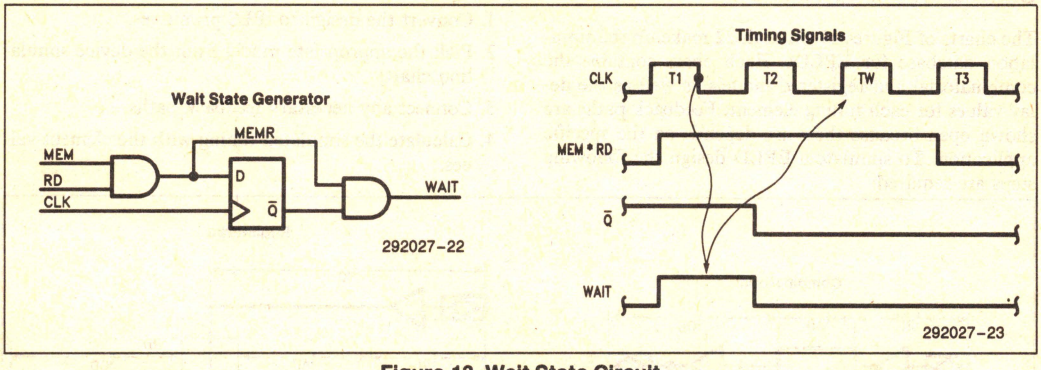


Figure 13. Wait State Circuit

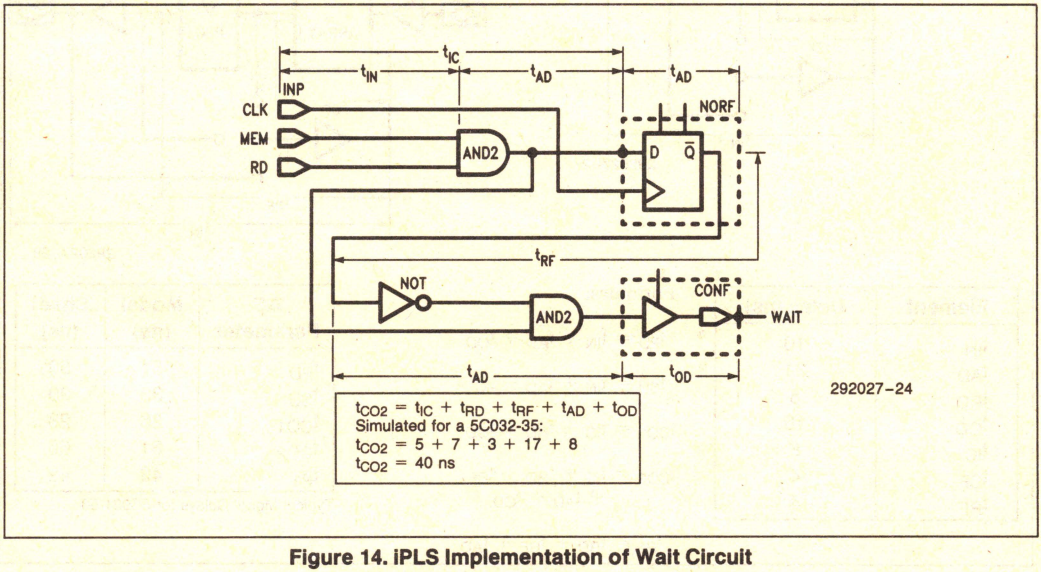


Figure 14. iPLS Implementation of Wait Circuit

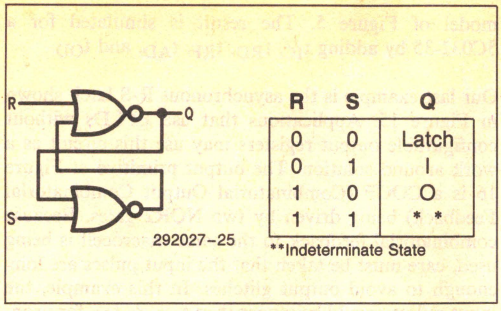


Figure 15. R-S Flip-Flop

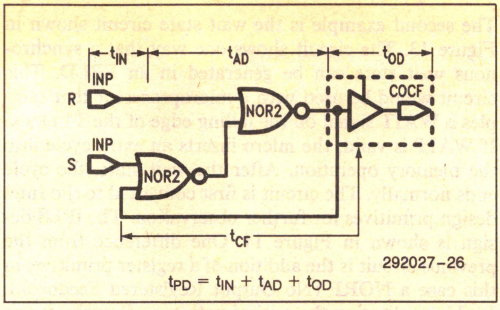
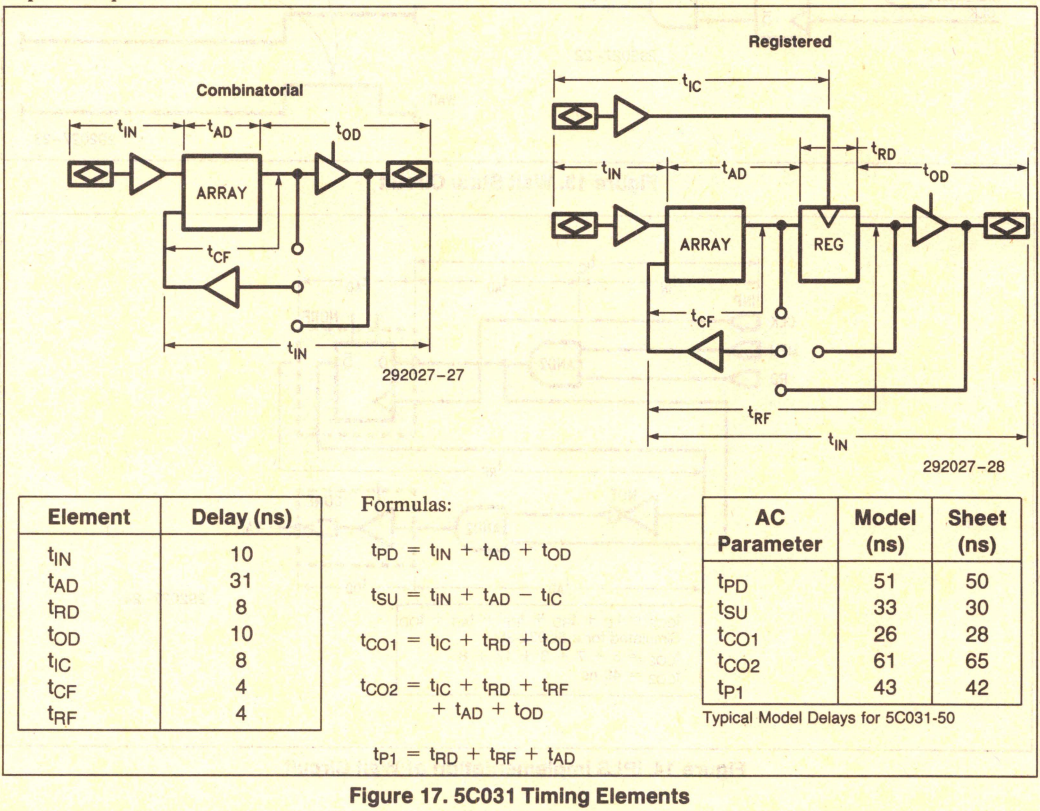


Figure 16. iPLS Implementation of R-S Flip-Flop

SIMULATION CHARTS

The charts of Figures 17 through 22 make up the simulation database for EPLDs. Each chart contains the combinatorial and registered models as well as the delay values for each timing element. Feedback paths are shown open because their use depends on the specific application. To simulate a EPLD design the following steps are required:

1. Convert the design to iPLS primitives.
2. Pick the appropriate model from the device simulation chart.
3. Connect any necessary feedback paths.
4. Calculate the simulated timing with the element values.



Element	Delay (ns)
t_{IN}	10
t_{AD}	31
t_{RD}	8
t_{OD}	10
t_{IC}	8
t_{CF}	4
t_{RF}	4

Formulas:

$t_{PD} = t_{IN} + t_{AD} + t_{OD}$

$t_{SU} = t_{IN} + t_{AD} - t_{IC}$

$t_{CO1} = t_{IC} + t_{RD} + t_{OD}$

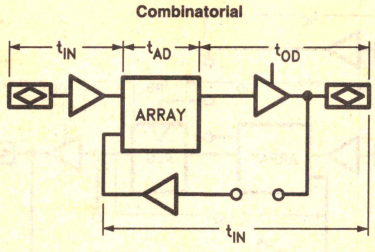
$t_{CO2} = t_{IC} + t_{RD} + t_{RF} + t_{AD} + t_{OD}$

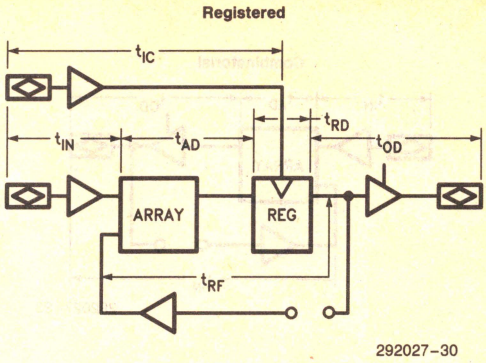
$t_{P1} = t_{RD} + t_{RF} + t_{AD}$

AC Parameter	Model (ns)	Sheet (ns)
t_{PD}	51	50
t_{SU}	33	30
t_{CO1}	26	28
t_{CO2}	61	65
t_{P1}	43	42

Typical Model Delays for 5C031-50

Figure 17. 5C031 Timing Elements





Element	Delay (ns)
t_{IN}	8
t_{AD}	17
t_{RD}	7
t_{OD}	8
t_{IC}	5
t_{CF}	3
t_{RF}	3

Formulas:

$t_{PD} = t_{IN} + t_{AD} + t_{OD}$

$t_{SU} = t_{IN} + t_{AD} - t_{IC}$

$t_{CO1} = t_{IC} + t_{RD} + t_{OD}$

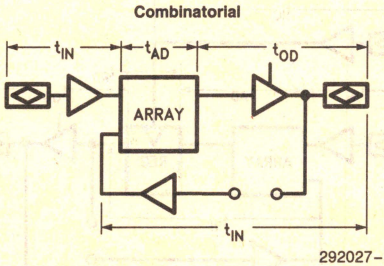
$t_{CO2} = t_{IC} + t_{RD} + t_{RF} + t_{AD} + t_{OD}$

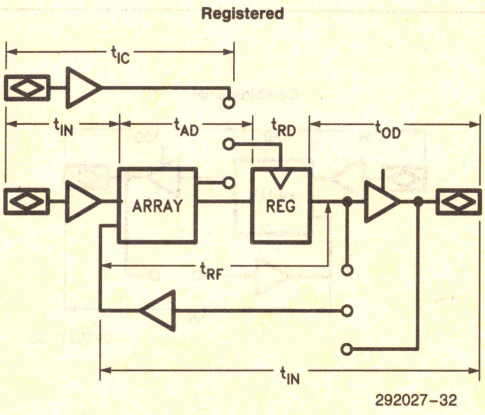
$t_{P1} = t_{RD} + t_{RF} + t_{AD}$

AC Parameter	Model (ns)	Sheet (ns)
t_{PD}	33	35
t_{SU}	20	25
t_{CO1}	20	20
t_{CO2}	40	42
t_{P1}	27	30

Typical Model Delays for 5C032-35

Figure 18. 5C032 Timing Elements





Element	Delay (ns)
t_{IN}	10
t_{AD}	30
t_{RD}	7
t_{OD}	10
t_{IC}	8
t_{CF}	5
t_{RF}	5

Formulas:

$t_{PD} = t_{IN} + t_{AD} + t_{OD}$

$t_{SU} = t_{IN} + t_{AD} - t_{IC}$

$t_{CO1} = t_{IC} + t_{RD} + t_{OD}$

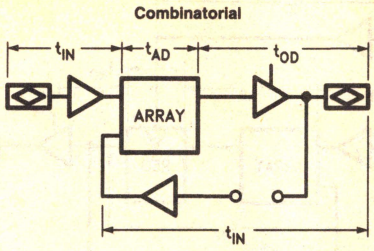
$t_{CO2} = t_{IC} + t_{RD} + t_{RF} + t_{AD} + t_{OD}$

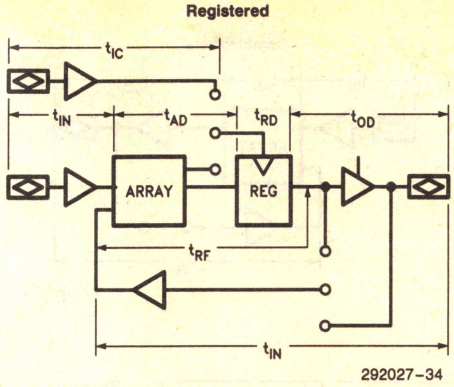
$t_{P1} = t_{RD} + t_{RF} + t_{AD}$

AC Parameter	Model (ns)	Sheet (ns)
t_{PD}	50	55
t_{SU}	32	35
t_{CO1}	25	25
t_{CO2}	62	60
t_{P1}	42	55

Typical Model Delays for 5C060-55

Figure 19. 5C060 Timing Elements





Element	Delay (ns)
t_{IN}	10
t_{AD}	37
t_{RD}	7
t_{OD}	10
t_{IC}	7
t_{CF}	5
t_{RF}	5

Formulas:

$t_{PD} = t_{IN} + t_{AD} + t_{OD}$

$t_{SU} = t_{IN} + t_{AD} - t_{IC}$

$t_{CO1} = t_{IC} + t_{RD} + t_{OD}$

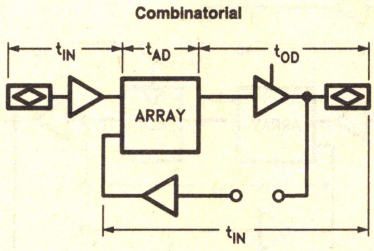
$t_{CO2} = t_{IC} + t_{RD} + t_{RF} + t_{AD} + t_{OD}$

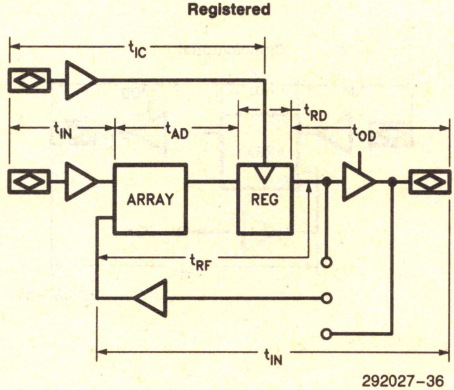
$t_{P1} = t_{RD} + t_{RF} + t_{AD}$

AC Parameter	Model (ns)	Sheet (ns)
t_{PD}	57	50
t_{SU}	40	45
t_{CO1}	24	20
t_{CO2}	61	55
t_{P1}	49	50

Typical Model Delays for 5C090-45

Figure 20. 5C090 Timing Elements





Element	Delay (ns)
t_{IN}	12
t_{AD}	44
t_{RD}	7
t_{OD}	12
t_{IC}	16
t_{CF}	5
t_{RF}	5

Formulas:

$t_{PD} = t_{IN} + t_{AD} + t_{OD}$

$t_{SU} = t_{IN} + t_{AD} - t_{IC}$

$t_{CO1} = t_{IC} + t_{RD} + t_{OD}$

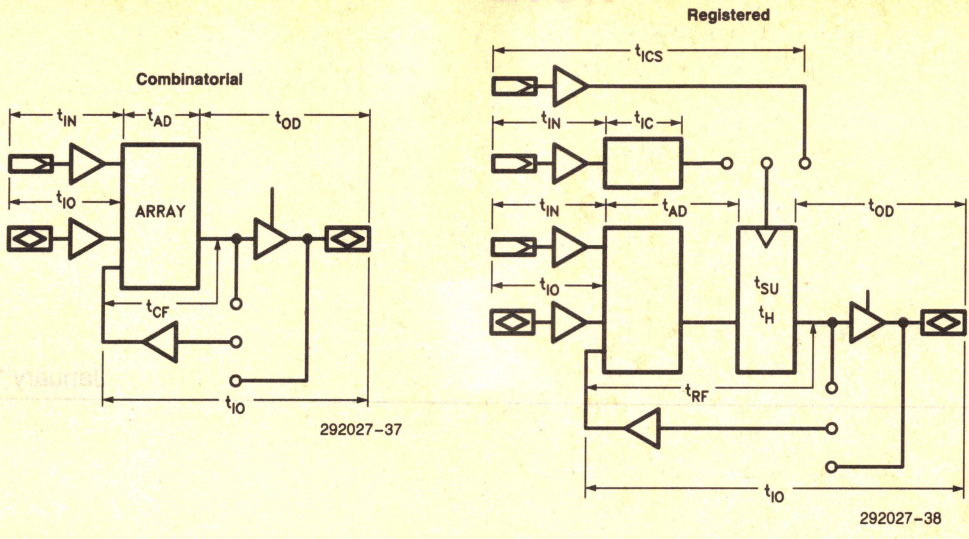
$t_{CO2} = t_{IC} + t_{RD} + t_{RF} + t_{AD} + t_{OD}$

$t_{P1} = t_{RD} + t_{RF} + t_{AD}$

AC Parameter	Model (ns)	Sheet (ns)
t_{PD}	68	65
t_{SU}	40	47
t_{CO1}	35	33
t_{CO2}	80	75
t_{P1}	56	55

Typical Model Delays for 5C121-65

Figure 21. 5C121 Timing Elements



Element	Delay (ns)
t_{IN}	12
t_{IO}	20
t_{AD}	44
t_{OD}	19
t_{IC}	30
t_{ICS}	3
t_{CF}	5
t_{RF}	10
t_{SU}	14
t_H	13

Formulas:

$t_{PD} = t_{IN} + t_{AD} + t_{OD}$

$t_{SU} = t_{IN} + t_{AD} - t_{ICS}$

$t_{CO1} = t_{ICS} + t_H + t_{OD}$

$t_{CO2} = t_{ICS} + t_H + t_{AD} + t_{OD}$

$t_{P1} = t_H + t_{RF} + t_{AD}$

AC Parameter	Model (ns)	Sheet (ns)
t_{PD}	75	75
t_{SU}	53	56
t_{CO1}	35	30
t_{CO2}	79	82
t_{P1}	67	65

Typical Model Delays for 5C180-75

Figure 22. 5C180 Timing Elements

SUMMARY

Timing simulation provides a way to verify a delay path without resorting to external measurements or breadboarding a design. Simulation of EPLDs can be done by implementing the design with the iPLS design primitives, modeling the device, and using the simulation charts. By applying these concepts, the engineer can simulate EPLD designs incorporated in one or more EPLDs.

REFERENCES

1. Intel User Defined Logic Handbook. EPLD Volume.
2. Intel Components Quality/Reliability Handbook.
3. Intel Programmable Logic Software User Guide.

APPLICATION NOTE

AP-307

January 1987

EPLDs, PLAs and TTL Comparing the "Hidden Costs" in Production

PEDRO VARGAS
PROGRAMMABLE LOGIC APPLICATIONS
INTEL CORPORATION

Order Number: 292030-001

INTRODUCTION

When comparing logic alternatives, too often the outcome is dominated by the piece price of the components. A side by side comparison based on component costs only, may give the appearance that EPLDs are cost prohibitive. However, when the overall cost of manufacturing a system is considered, the higher integration of EPLDs proves to be a cost-effective solution.

OBJECTIVE

This application note examines the total costs associated with designing, prototyping, and manufacturing a system. Once these costs have been examined, a comparison is made between EPLDs and other logic alternatives. By being aware of these additional costs, the engineer can make a more accurate cost comparison as a design is begun.

COSTS DEFINED

Costs can be difficult to pinpoint, let alone measure. However, with a bit of examination, we can break down costs into the following categories;

- Design costs — the cost of conceiving a product
- Prototype costs — first implementation of the product idea
- Production costs — volume manufacturing of the product

Usually, the brunt of the cost for the first two categories is dismissed as NRE (non recurring expense). The effect of these costs on the overall project is examined later, let's look at the third category. Production costs, can be further broken down into;

- Component costs — the cost of the parts per board
- Inspection costs — labor costs for receiving the parts
- Inventory costs — the cost for storing, handling and dispensing the parts
- PCB fabrication — the cost for labor and equipment used in building a board
- Integration costs — the cost of harnesses, enclosures, nuts and bolts etc.

It's important to understand how the cost of a product is affected not only by the cost of the ICs used, but also by the other costs listed above. Figure 1 is a graph which shows this relationship.

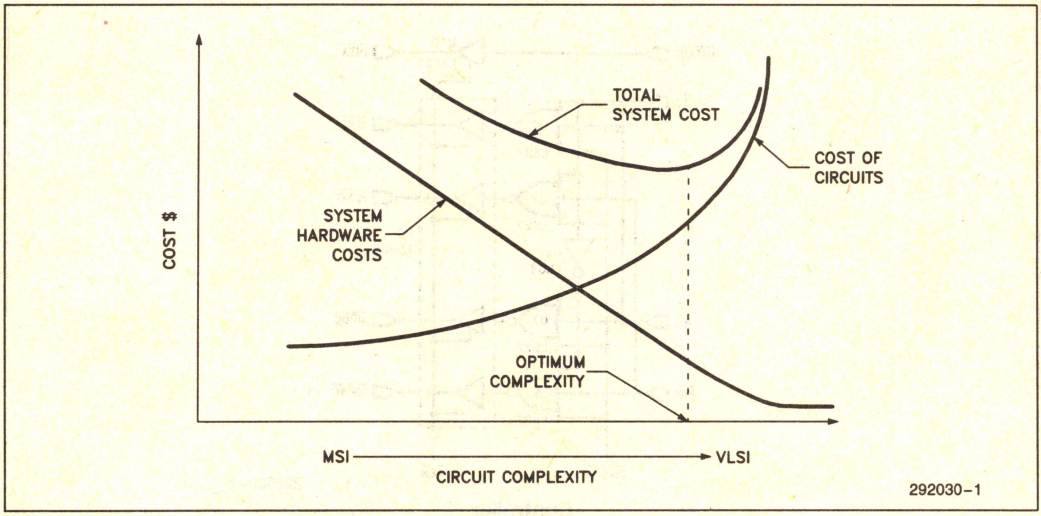


Figure 1. Optimizing Circuit Complexity

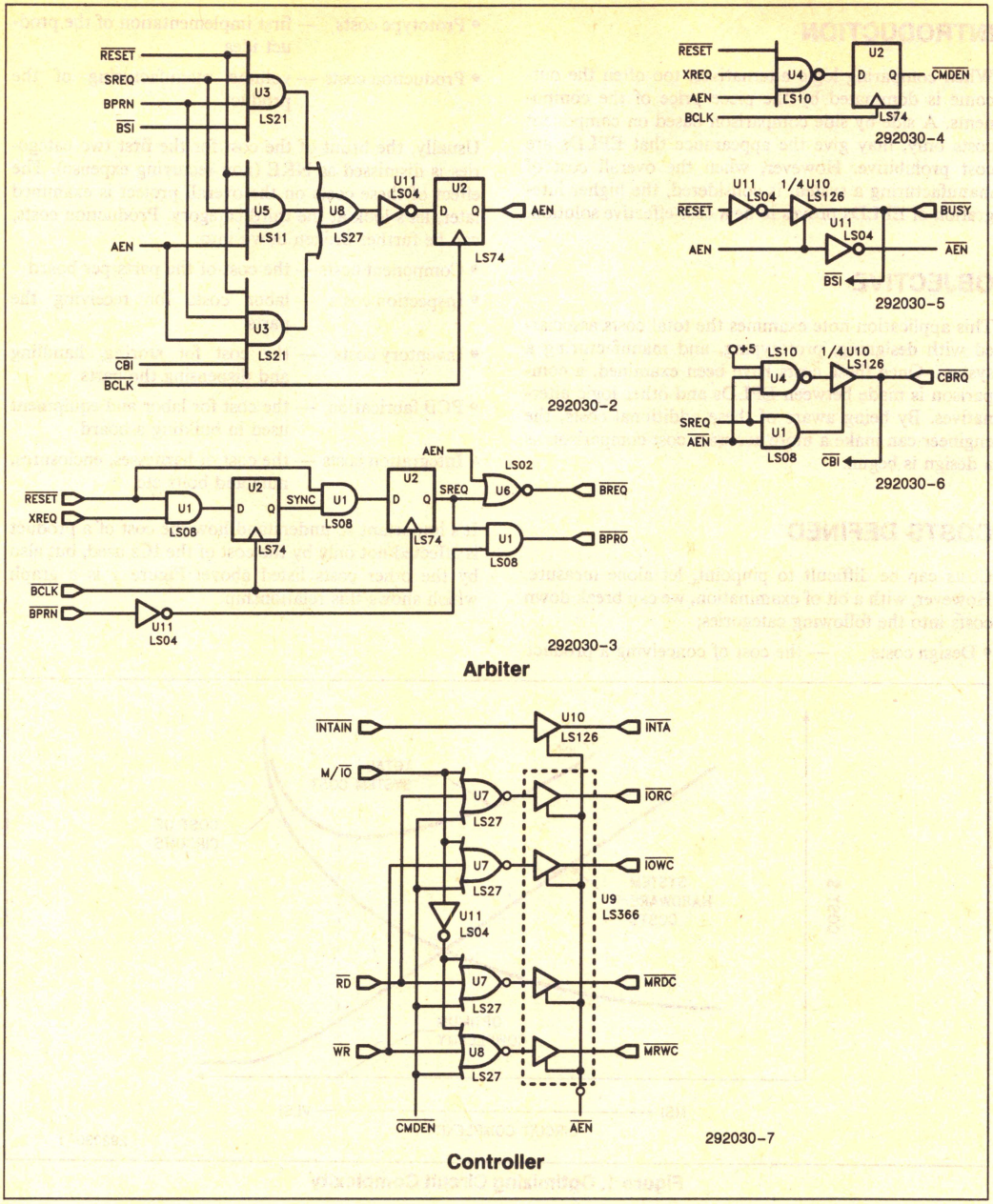


Figure 2. MULTIBUS Arbiter/Controller–TTL Implementation

The graph shows that as the density of the components used in a system progresses from SSI to VLSI, the cost for these devices increases. This isn't surprising, denser chips cost more to make. At the same time, by using denser devices, system hardware cost decreases. This is shown by the center line, which encompasses all the costs listed above. The bathtub curve above these shows the effect that denser ICs has on a system. That is, by using higher integration ICs, more functions are removed from the board. This in turn reduces the cost of the system in labor and parts costs.

A cost-effective product is one that uses the most efficient logic for the application. It's important to note that use of the least expensive component may not translate into system cost savings.

PAL® is a registered trademark of Monolithic Memories Inc.

ARBITER CIRCUIT

Let's explore costs in more detail with an example. The example used here is the circuit of Figure 2, a MULTIBUS® I arbiter/controller. The circuit is used by bus masters arbitrating for control of the bus. Our implementation comparison contrasts TTL, PAL®, and EPLD solutions.

Implementation Requirements

The TTL implementation is typical of many board level designs in the sense that it relies on inexpensive LSTTL. Figure 2 shows that the implementation is composed of standard logic gates and D-latches. The component list in Table 1 shows the circuit breakdown in more detail.[20]

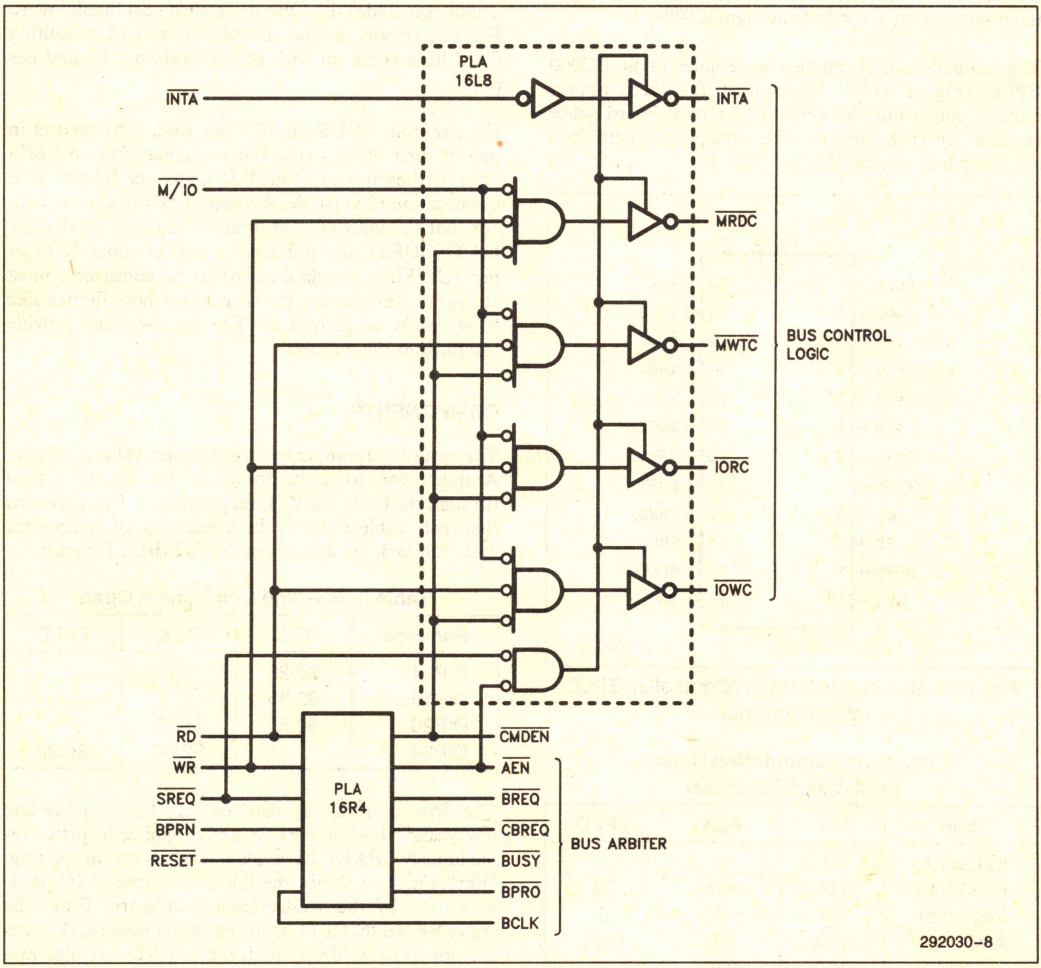


Figure 3. MULTIBUS Arbiter/Controller-PAL Implementation

Table 1. Arbiter/Controller TTL Component List

IC	Type	DIP	I_{CC} (mA)	Area (in^2)	Cost $
U1	LS08	14 PIN	8.8	0.21	0.18
U2	LS74	14 PIN	8	0.21	0.24
U3	LS21	14 PIN	4.4	0.21	0.22
U4	LS10	14 PIN	3.3	0.21	0.16
U5	LS11	14 PIN	6.6	0.21	0.22
U6	LS02	14 PIN	5.4	0.21	0.17
U7	LS27	14 PIN	6.8	0.21	0.23
U8	LS27	14 PIN	6.8	0.21	0.23
U9	LS366	16 PIN	21	0.24	0.39
U10	LS126	14 PIN	22	0.21	0.39
U11	LS04	14 PIN	6.6	0.21	0.16

The PAL version of the circuit is shown in Figure 3. Two PALs are used due to the requirement of registered outputs on several of the signals.[20]

The complete circuit can also be designed in one 5C060 EPLD (Figure 4).[18] Looking at the three figures quickly points out the amount of circuit board space required by each version. The three implementations are compared side by side in Table 2.

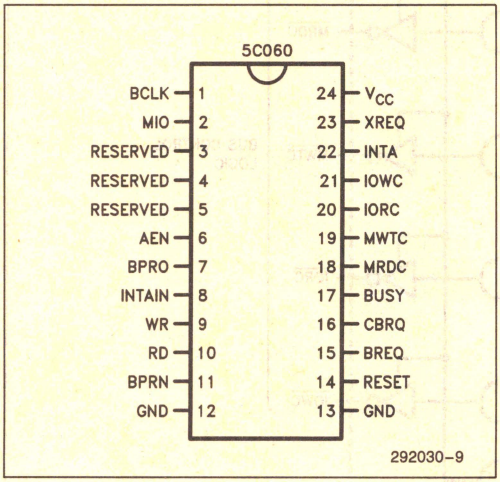


Figure 4. MULTIBUS Arbiter/Controller-EPLD Implementation

Table 2. Implementation Results for Arbiter/Controller

Item	TTL	PLA	EPLD
IC Count	11	2	1
Pin Count	156	40	24
Interconn	36	7	0
Area	2.34	0.6	0.36
I_{CC} (mA)	100	240	15
P_{wr} (mW)	500	1,200	75

- IC Count — The total chip count
- Pin Count — The total number of IC pins
- Interconnections — The traces required to connect logic gates together
- Area (inches-square) — The sum of the area of all ICs
- I_{CC} (mA) — The current consumed while active
- P_{wr} (mW) — Total power consumption at 5 VDC.

Production Costs

Earlier, we noted that production costs consist of many variables. Usually, these variables are lumped together under the term "hidden cost". Although hidden costs are kept in mind by engineers, lack of tangible figures usually precludes their use in detailed cost breakdowns. For this reason, several manufacturers and consulting firms have come up with typical costs per IC and per pin.

For example, SOURCE III (San Jose, CA) reports in one of their studies that the manufacturing cost of a system translates to about 0.35 cents per IC pin. ICE Corporation (Scottsdale, AZ) and EDN magazine concur that the inserted cost of an IC is about $2 dollars. DATAQUEST also published a cost of about $2 to $4 per IC. While the data seems to be consistent, most engineers want to see for themselves how figures like these might be arrived at. The next sections provide insight into this process.

COMPONENTS

The cost of the component is the easiest value to obtain. A quick call to a distributer or (at worst) a scan through the back of BYTE magazine (for TTL) gives us this cost. Table 3 shows the breakdown of component costs for each version of our MULTIBUS I circuit.

Table 3. Average Component Costs

Package	TTL	PLA	EPLD
DIP14	$0.25		
DIP16	$0.35		
DIP20	$0.55	$1.50	
DIP24		$2.90	$6.00

The price of TTL has changed very little for the last few years[24] while EPLDs are dropping in price tremendously. PALs have also leveled off in pricing. Why? Figure 5 shows the life cycle curve of IC products used by the semiconductor industry. From the curve we see that TTL is in the stable range and prices are not likely to drop much more. PALs are also maturing and approaching a stable pricing range. EPLDs however, are in a growth area and historically this is

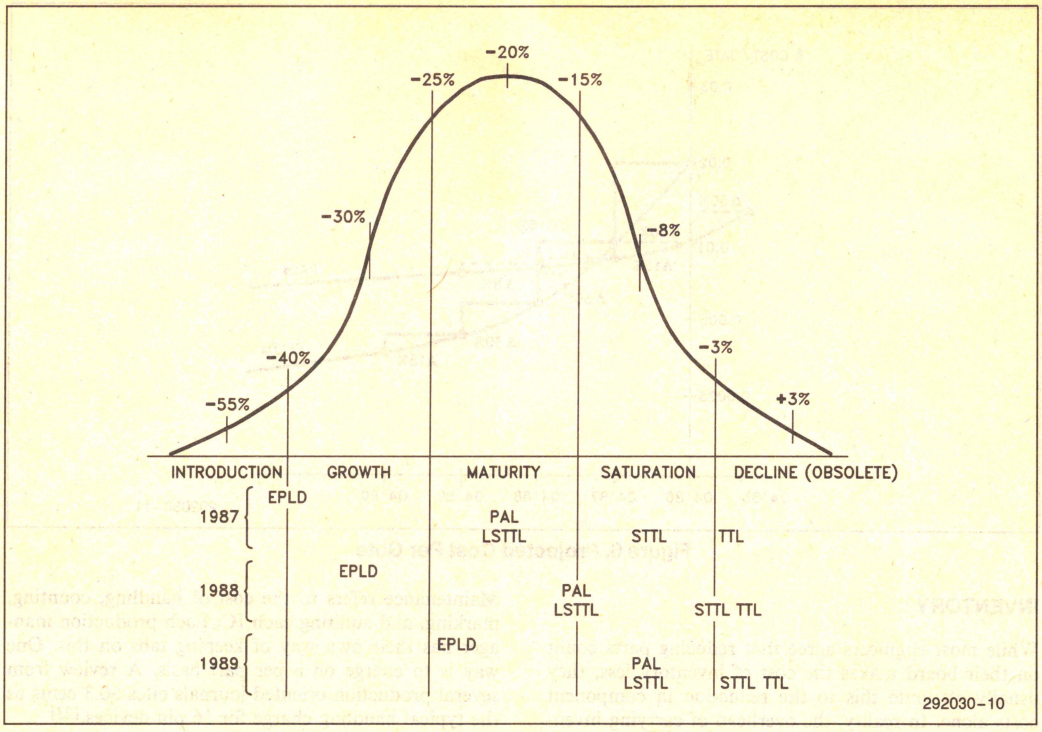


Figure 5. Typical Price Changes Through Semiconductor Product Life Cycle

where the heaviest pricing pressure is. This means that while EPLDs might be expensive (per part) right now, it's not out of the question to expect a 30% per year price reduction as the process is honed and perfected. In other words, it's also important to consider the price of a component at the projected production date, not just at design time.

Life cycle position is also important in understanding the gate cost that is associated with programmable logic devices like PALs and EPLDs. This relationship is shown in Figure 6. The curves translate our observation that newer devices have steeper price cuts during their introduction phase. The PAL curve shows that the cost per gate is leveling off due to the maturity of the device. In contrast, the EPLD is in the growth region, and based on the traditional price reductions, shows a cost per gate that intersects and bypasses the PAL curve.

INCOMING INSPECTION

For most companies, incoming inspection is more than taking the parts and putting them on the shelf. Most have visual checking as well as some form of IC testing. The variables here are, what amount of human intervention is needed, are automatic handlers needed, are "go/no go" tests or "binning" done automatically? The typical scenario means that components are graded and tested individually, and then placed into one of several bins or kitted. Because the operators handle a large variety of pinned devices (resistors, capacitors, ICs), the cost can be distributed on a per pin basis. Many companies use a penny per pin for this cost.[16]

Inspection cost = $0.01 per pin

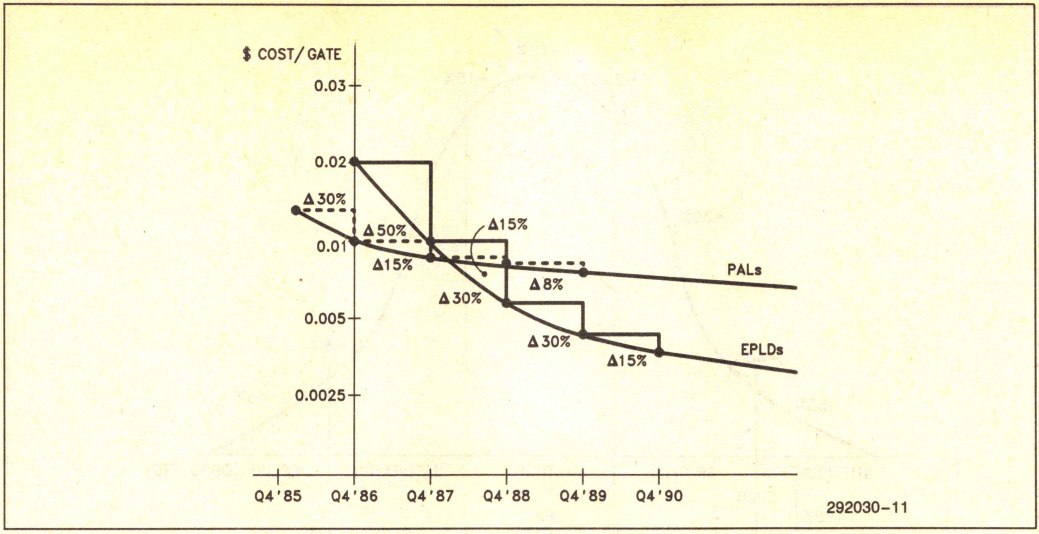


Figure 6. Projected Cost Per Gate

INVENTORY

While most engineers agree that reducing parts count on their board makes the cost of inventory less, they usually attribute this to the reduction in component costs alone. In reality, the overhead of carrying inventory is made up of the following factors;[21]

- Cost of the component
- Cost of storage
- Maintenance costs
- Data processing
- Usage
- Taxes insurance and interest
- Turnover rate

The American Production and Inventory Control Society (APICS) reports that since 1973 the median cost of carrying inventory has been about 25% of total production costs. They also note that the largest contributing factors are the cost of materials handling storage, and data processing. For simplicity, let's limit our inventory cost to these items.

Inventory cost = storage + maintenance + processing

Depending on the locale of a company, the cost of storage can vary greatly. However, this cost is charged on a square foot per year basis. Lets assume a conservative figure of $20 dollars and distribute this amoung the ICs in our example circuit.

storage = [Total IC area (sq. ft.) × $20]/IC count

Maintenance refers to the cost of handling, counting, marking, and auditing each IC. Each production manager has their own way of keeping tabs on this. One way is to charge on a per part basis. A review from several production oriented journals cites $0.3 cents as the typical handling charge for 16 pin devices.[23]

Maintenance = $0.03 per 16 pin part.

Processing[21] usually entails a parts log that tracks each part by manufacturer, cost, second source etc. Also, monthly shortage reports are quite common as are quarterly orders and audits. Limiting this cost to paper only, at one sheet of paper per week, per year, at a cost of a penny per part type;

Processing = $0.52 per part type per year

PCB FABRICATION

The cost of manufacturing (cutting, etching, drilling) a circuit board seems to vary around two pricing methods. Some fab houses charge on a square inch basis. Others base their price on a gut feeling based on previous jobs. The square inch method is the most common.

Items of interest in evaluating PCB costs are, number of ICs, number of traces and vias, and in general, the complexity of the board. Traces that are smaller than 10 mils require extra care in etching. Depending on complexity, and additional charge might be added to the area cost. This charge covers material loss in case of low etch yields. Yield is directly dependent on the number of ICs on a board. In other words, more ICs mean more holes, tighter traces, and a greater chance of losing some boards in their processing. The average going

rate is $0.20 cents per inch for double-sided boards. The price increases by about 40% for every two layers. This extra charge, however is too subjective to consider in our comparison.

PCB Fab = [$0.20 × total IC area (sq. inch)]/IC count

Traces

There is a real cost involved with traces, which doesn't surface until later in the production cycle or on a later board revision. A technical paper presented at the 1984 international Test Conference[1] estimates that the cost of a trace on a board is ten to thirty times that of one made in silicon. The cost of traces is taken up by:

- Increased drilling (more traces = more vias = more holes)
- Lower PCB yield (smaller mill lines drop the board yield)
- Increased risk of trace to trace shorts (lower reliability)
- More expensive artwork mods (it costs more to move traces around on a board)
- More expensive PCB mods (cost of cuts, jumpers, and rework)

In our circuit example, an extra trace is that which is unnecessary in contrasting implementations. For example, referring to Figure 2, of all the traces required to connect/RESET in the TTL implementation, only one will be required for the EPLD and PAL circuit (the input); the others won't be needed.

For our comparison, let's take the median value of twenty as our multiplying factor. Since a silicon trace costs an order of magnitude less than an EPLD gate ($0.01), the resulting cost of a PCB trace is;

($0.01/10) × 20 = $0.02 cents per trace

Trace cost = [total trace count × $0.02]/IC count

ASSEMBLY

The cost of assembling a board is largely dependent on labor charges and capital. Assembly consists of lead forming, component insertion, and soldering. The labor charge is hourly and varies between domestic and offshore assembly houses. While machines can certainly do lead cutting, crimping, and insertion, human intervention is still an expensive presence. Assembly costs can be charged on a per board or per chip basis. The latter is more appropriate for our comparison. The average charge (domestically) is about $0.10 per IC.

Assembly = $0.10 per 16 pin part

One important result of using high integration parts like EPLDs is that the assembly procedures (manual or automatic) go smoother. This is due to fewer parts being handled, and less overheating of the equipment. Overall, the industry reports less insertion faults (parts stuffed wrong) as denser ICs are used and as insertion equipment matures with them.

TEST

Test strategies can vary, but the typical test flow for a board[3] is shown in Figure 7. The process is basically taking a board through increasing complexity levels of testing. For example, ATE might be a bed of nails fixture that catches 60 percent of the faults. Test bed is usually a backplane with all boards known good except for the one under test. System test is the final integration of all the boards that were tested individually.

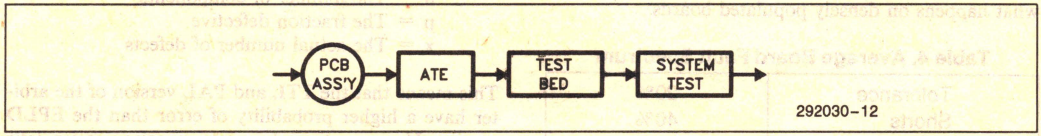


Figure 7. Typical Test Flow

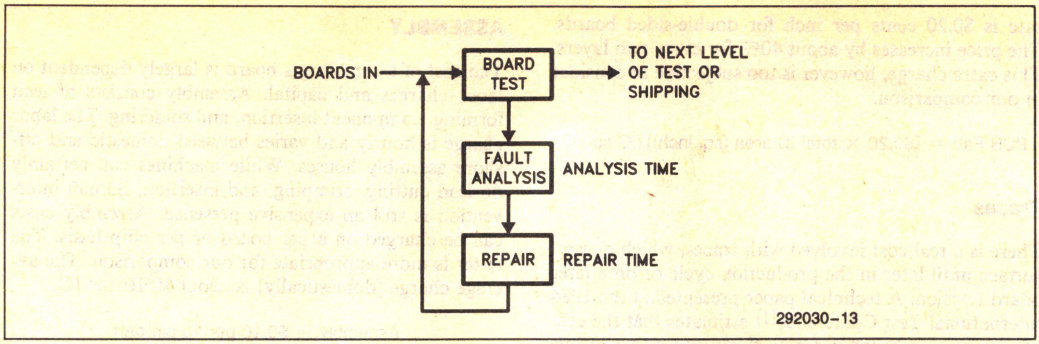


Figure 8. Typical Test and Repair Loop

Errors can occur at any step of the test flow; each time this happens, a test loop is initiated. This loop is depicted in Figure 8. The cost for testing a device depends on the cost of the equipment, depreciation, the labor rate, and other factors that are company dependent. There are several ways to reduce test costs, but the best way is to reduce the probability of errors occuring. There is no question that as the number of ICs increases, so does the probability of error.

With all things considered, the industry reports a nominal test cost of about $0.15 per IC.[27][28]

Test cost = $0.15 per 16 pin IC

REWORK

The cost of rework is best understood by considering the cause of errors in more detail. Errors are typically caused by poor board quality, inadequate solder process, tolerance of insertion, and of course, bad chips. Table 4 shows the average board fault spectrum. The figures are a conclusion reached by EVALUATION ENGINEERING magazine[10] as to what the industry is currently seeing. The table shows that the majority of board errors is due to solder shorts. These errors are the result of traces or IC holes being too close, which is what happens on densely populated boards.

Table 4. Average Board Fault Spectrum

Tolerance	20%
Shorts	40%
Insertion	30%
Bad Parts	10%

Of all the material costs associated with rework, the main cost is the time spent on a repair. Considering that it takes approximately two minutes to desolder, insert, resolder, and clean a component pin[9], one can see that more ICs on a board directly affect cost. Repair times also increase dramatically on multi-layer boards that might have been doubled sided if denser logic was used.

For our comparison, let's assume that our test equipment is 95% efficient in finding solder faults on the first pass (no loop). This leaves 5% of the faults that go undetected and eventually must be found and repaired. The estimated cost per pin based on a $6.00 hourly wage and the two minute repair time is approximately $0.02 cents.

Rework = [$0.02 × total pin count]/IC count

It is important to note that the probability of errors is based on a Poisson distribution[8] that increases exponentially with the number of pins and components. This distribution is used in wave solder processing to correct for solder errors. Mathematically this is expressed as:

$$P = \frac{e^{-np}(np)^x}{X!}$$

where; P = The probability that a defect will occur
n = The number of components
p = The fraction defective
x = The actual number of defects

This means that the TTL and PAL version of the arbiter have a higher probability of error than the EPLD version. However, to make our comparison easier, let's simplify this to more of a linear relation. For each implementation, the rework cost per IC is calculated by;

Rework cost = [(total pin count) × (5%) × ($0.02 cents)]/IC count

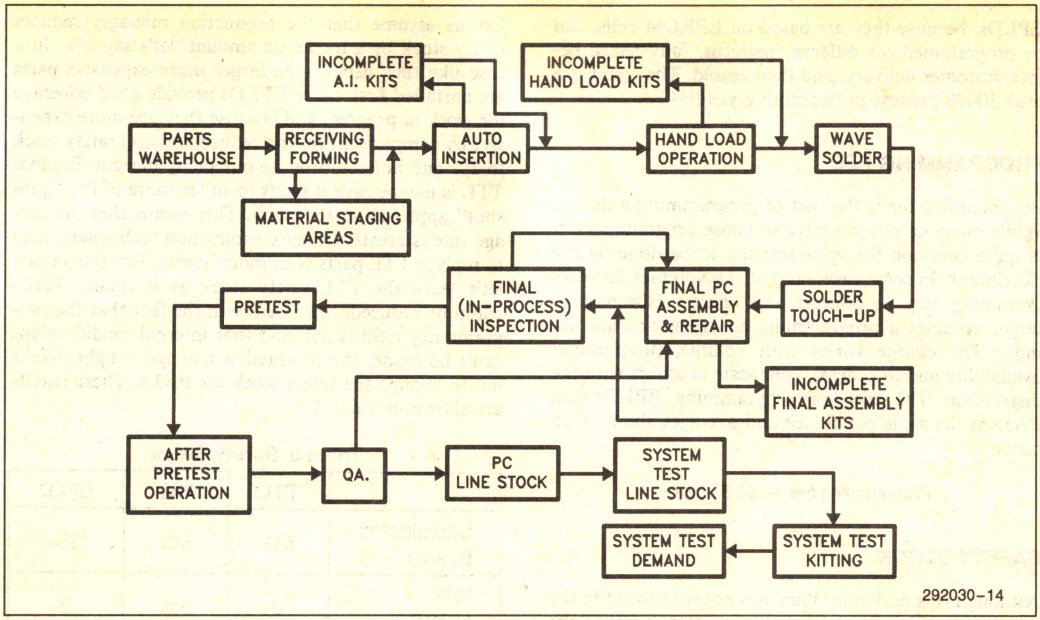


Figure 9. Example of a Production Line

QUALITY CONTROL

In most production operations, boards go through several steps of quality inspection. The bare board might be inspected after preliminary tests and after system tests. Although 100% inspection should theoretically eliminate all errors, in real life this rarely happens. The main reason for this is the complexity of the production and rework loops as shown in Figure 9.

Quality control's purpose is to remove defective products and either junk them or rework them, neither of which is cost effective. The best approach is to design the quality in, not fix it in. One way to design in quality is by reducing the possibility of errors and increasing the reliability of a product. This is one of the primary advantages of dense logic (like EPLDs and PALs) over TTL.

A survey conducted by *CIRCUITS MANUFACTURING* magazine[8] yielded the cost of $10 to $50 dollars to inspect, find, and repair a defect on a board. They summarized that the actual cost of inspection is about $0.004 for each hole on a board. With this in mind, let us assume a 100% inspection of our arbiter circuit for each implementation. This means that each pin (and every trace via) will have to be looked at. The calculation for this is;

$$\text{QC cost} = (\text{total pin count} \times \$0.004)/\text{IC count}$$

POWER SUPPLY

Price for 5V, single output, switching power supplies as advertised by several vendors is $1.00 per watt. The calculation for determining power supply costs in our comparison is:

$$\text{Power cost} = [(5\text{VDC} \times I_{CC}\ (\text{mA})) \times \$1.00\ \text{per watt}]/\text{IC count}$$

Additional Costs

In addition to the more obvious costs, there are several other items that contribute to the "hidden cost" of a system.

PROGRAMMING LOSS

Because PALs are a one time programmable type of device, full testing can't be done on them without destroying the user's fuses. For this reason PALs have a published programming loss of 2%[20]. The cost for this is:

$$\text{Programming loss} = (\text{PAL IC count} \times 0.02) \times \text{PAL cost per IC}$$

EPLDs, because they are based on EPROM cells, can be programmed for different patterns, fully tested before customer delivery, and then erased. The result is a near 100% percent programming yield[22].

PROGRAMMING FEE

Programming fee is the cost of programming a device. While many companies have in-house programmers, it is quite common for programming to be done by the distributor. In some cases, and at low volumes, the programming may be done free of charge. However, at larger volumes a programming charge is not uncommon. The charge varies with volume, programmer availability and in general, your state of affairs with the distributor. The cost for programming EPLDs and PALs is the same per device and averages about $0.25 cents.

Programming fee = $0.25 cents

SAFETY STOCK

Although this particular item was not mentioned in the inventory section, it plays a very important role in the production world. Safety stock[21] is extra ICs ordered to cover for unexpected events. Unexpected here might be a large unforeseen customer order or simply a bad batch of parts.

While industry seems to strive for the optimum JIT (just in time) production[14][16], which stresses minimal inventory until needed, it's not unusual for production managers to carry a five to ten percent inventory buffer depending on the cost of the part. In most cases, the larger expensive parts like microprocessors, peripheral controllers, and other LSI devices are safety stocked in smaller quantities.

Let's assume that the safety stock is to be a maximum of 10%. Five percent might be used to cover for the unexpected occurrences, and five for WIP (work in process) modifications. Since all parts have the same probability of unexpected events we can assign that percentage equally. Justifying the second 5% depends on the IC technology itself. For instance, WIP modifications usually require cuts and jumpers on TTL, therefore it's unnecessary to order the additional 5%. In process modifications to an EPLD are done simply by reprogramming it, here again there is no need for the additional 5%. PALs however cannot be cut and jumpered (internally) nor can they be reprogrammed. Also, there is the possbility that "on the shelf" PALs will be programmed in advance, therefore a WIP mod that impacts their function means that those parts must be obsoleted (junked). In this case, an additional 5% is justifiable.

Let us assume that the production manager reduces safety stock by a moderate amount, let's say 3%. In a case like this, usually the larger more expensive parts are curtailed first. Since EPLDs provide good coverage for work in progress and because they are more expensive by comparison, we can reduce the total safety stock to 2% and not compromise our safety margin. Because TTL is inexpensive it tends to suffer more of the "gunshot" approach in testing[7]. This means that the useage rate is greater because production technicians tend to replace TTL parts with more liberty. For this reason let's leave the TTL safety stock as it stands. PALs could be reduced, but faced with the fact that the programming yield is 2% and that internal modifications can't be made, the production manager might decide not to change the safety stock for PALs. These results are shown in Table 5.

Table 5. Safety Stock

	TTL	PAL	EPLD
Unexpected Events	5%	5%	2%
WIP MODS	0	5%	0
Total	5%	10%	2%

The safety stock calculation for each implementation is:

Safety stock = (% of stock × IC type × IC type cost)/IC count

DE-COUPLING CAPACITORS

While adding caps solves many problems due to system noise, it also increases the cost of PCB layout, PCB fab, and adds an additional burden on all of our other costs. For a TTL system, a good de-coupling rule of thumb is to use one 0.01 μf per each synchronous driven gate and at least 0.1 μf per 20 gates regardless of synchronicity. Engineers recognize the need for decoupling and usually take it a step further by using one capacitor per IC. Most boards reflect this practice, which, in itself is very good. However, the addition of all these caps is definitely measurable, in both component and systems cost.

The average cost of a ceramic capacitor in moderate quantities is about half a cent. For our comparison we will follow the accepted practice and de-couple each TTL, PAL, and EPLD device. Our capacitor cost is then:

De-coupling cost = $0.005 × IC count

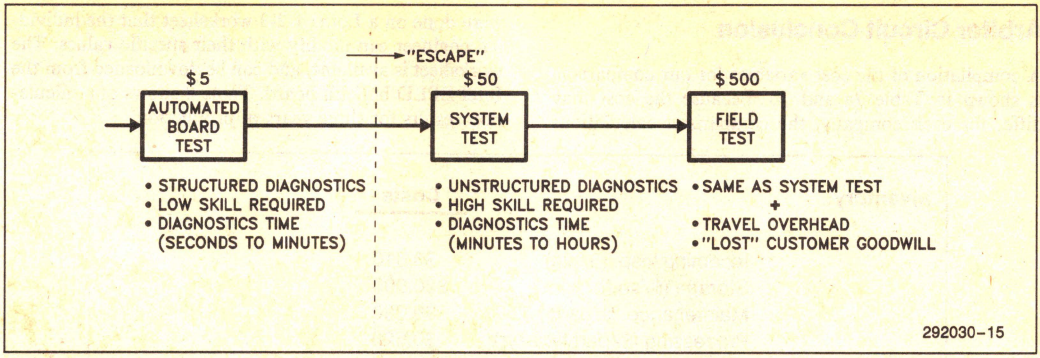


Figure 10. Escape Costs

Other Costs To Consider

Eventually, some place toward the end of a production line, a board becomes part of a system. At this point it is housed in an enclosure and all the necessary cabling is done. Even here, however, the impact of using a particular IC technology can still be felt.

DEFECT ESCAPES

One very significant item that the test community acknowledges is the cost of "escapes"[4]. "Escape" is defined as a fault that goes through the early stages of board test undetected. Figure 10 shows the escape relationship. An industry rule of thumb states that the cost to detect a fault increases by an order of magnitude at each stage. This means that if it costs $5 to find a fault at the board test level, that same fault might cost $50 at the system level and $500 at the field level. An important relationship to remember, is that the number of faults per board increases logarithmically, as the number of components on the board increases[6]. The cost of an "escape" is difficult to quantify, but generally, a board with a higher component count has a greater cost[2][8].

CABLES/WIRING HARNESS

When the number of components or the power requirements of a system are reduced, a reduction in cables and wiring is usually expected. The cost savings here is either in the elimination of cables (because more functions are condensed into an IC) or the reduction of cable gauge or length (because less power is required, in the case of EPLDs). Also, fewer cables means fewer cable ties, connector pins, and mounting hardware. While this is a subjective figure, lets assume that the distributed cost of system cables is $0.25 per IC.

$$\text{Cable cost} = \$0.25 \times \text{IC count}$$

ENCLOSURE

Certain applications require reduced packaging or enclosure size. In industrial control for example, each line might require a complete system to monitor it's operation. In a case like this, a large bulky box full of boards might not be appropriate. A good example of the benefits that high integration logic provide enclosures, is the third market versions of the popular PC. Many of these companies have fully compatible versions that fit on a single board. EPLDs and PALs are capable of providing a cost savings in this respect. However, while PALs approach the density requirements, their large power needs render them counterproductive to the low power specs of small systems. TTL is just not as effective as either PALs or EPLDs.

For our comparison let us assume the cost of enclosure per chip is $0.75. The calculation is:

$$\text{Enclosure cost} = \$0.75 \times \text{IC count}$$

Table 6 shows the cable and enclosure costs for the MULTIBUS I circuit. Although the results are based on assumed values, we can see that a larger IC count influences the burdened cost of the system. Our final comparison will not use these figures, but they should be considered.

Table 6. Other Production Costs for Multibus I Circuit

	TTL	PLA	EPLD
Wiring/harness	$2.750	$0.500	$0.250
Enclosure	$8.250	$1.500	$0.750

Arbiter Circuit Conclusion

A compilation of the cost variables for our comparison is shown in Table 7a and 7b. Because the cost may differ for each company, the comparison calculations were done on a Lotus 1-2-3 worksheet that the individual engineer can modify with their specific values. The worksheet is available, and can be downloaded from the Intel EPLD bulletin board. Table 8 shows our calculation results for three years of production.

Inventory:	Costs
Incoming insp. ($/pin)	$0.010
Storage ($/sq.ft./yr)	$20.000
Maintenance ($/part)	$0.030
Processing ($/part type/yr)	$0.520
Safety stock (%)	2%

Manufacturing:	Costs
PCB fab. ($/sq.in.)	$0.200
Assembly ($/part)	$0.100
Test ($/part)	$0.150
Rework ($/pin)	$0.020
QC ($/pin)	$0.004
Power ($/watt)	$1.000
Interconn	$0.020
Program ($/part)	$0.250
Caps. (each)	$0.005

(a)

Integrated Circuits

Component Count:

Package	TTL	PLA	EPLD
DIP14	10		
DIP16	1		
DIP20	0	2	
DIP24			1

ICs	Types
TTL	10
PLA	2
EPLD	1

Circuit Requirements:	I_{CC} (max)	Interconnects
TTL circuit (total mA).	100	36
PLA circuit (total mA).	240	7
EPLD circuit (total mA).	15	0

(b)

Tables 7a and b. Multibus Arbiter/Controller Cost Variables

Table 8. MULTIBUS I Arbiter/Controller Production Costs

AVERAGE COMPONENT COST									
	Year 1			Year 2			Year 3		
Package	TTL	PLA	EPLD	TTL	PLA	EPLD	TTL	PLA	EPLD
DIP14	$0.25			$0.20			$0.19		
DIP16	$0.35			$0.30			$0.27		
DIP20	$0.55	$2.00		$0.38	$1.70		$0.35	$1.56	
DIP24			$6.00			$4.20			$2.90
PRODUCTION COSTS									
	Year 1			Year 2			Year 3		
Item (costs per part)	TTL	PLA	EPLD	TTL	PLA	EPLD	TTL	PLA	EPLD
Components	$0.259	$2.000	$6.000	$0.209	$1.700	$4.200	$0.197	$1.560	$2.900
Incoming Insp.	$0.142	$0.200	$0.240	$0.142	$0.200	$0.240	$0.142	$0.200	$0.240
Inventory									
Maintenance	$0.027	$0.038	$0.045	$0.027	$0.038	$0.045	$0.027	$0.038	$0.045
Storage	$0.030	$0.042	$0.050	$0.030	$0.042	$0.050	$0.030	$0.042	$0.050
Processing	$0.473	$0.520	$0.520	$0.473	$0.520	$0.520	$0.473	$0.520	$0.520
Printed Circuit Board									
Fabrication	$0.043	$0.060	$0.072	$0.043	$0.060	$0.072	$0.043	$0.060	$0.072
Trace costs	$0.065	$0.070	$0.000	$0.065	$0.070	$0.000	$0.065	$0.070	$0.000
Assembly	$0.089	$0.125	$0.150	$0.089	$0.125	$0.150	$0.089	$0.125	$0.150
Board test	$0.150	$0.150	$0.150	$0.150	$0.150	$0.150	$0.150	$0.150	$0.150
Rework	$0.014	$0.020	$0.024	$0.014	$0.020	$0.024	$0.014	$0.020	$0.024
QC	$0.057	$0.080	$0.096	$0.057	$0.080	$0.096	$0.057	$0.080	$0.096
Power Supply	$0.045	$0.600	$0.075	$0.045	$0.600	$0.075	$0.045	$0.600	$0.075
Total Cost/Part	$1.393	$3.904	$7.422	$1.343	$3.604	$5.622	$1.331	$3.464	$4.322
Total Cost/System	$15.321	$7.808	$7.422	$14.771	$7.208	$5.622	$14.641	$6.928	$4.322
Additional Costs/System									
Programming loss	$0.000	$0.080	$0.000	$0.000	$0.068	$0.000	$0.000	$0.062	$0.000
Safety stock	$0.143	$0.400	$0.120	$0.115	$0.340	$0.084	$0.109	$0.312	$0.058
Programming fee	$0.000	$0.500	$0.250	$0.000	$0.500	$0.250	$0.000	$0.500	$0.250
De-coupling caps	$0.055	$0.010	$0.005	$0.055	$0.010	$0.005	$0.055	$0.010	$0.005
True mfg. cost/system	$15.518	$8.798	$7.797	$14.941	$8.126	$5.961	$14.804	$7.813	$4.635

The comparison in component costs shows that the EPLD costs more than either a TTL or PAL IC. As costs are added, the figures for TTL and PALs begin to approach the cost of an EPLD. These are shown on the line labeled "Total cost/part".

The "Total cost/system" line shows the actual cost when all the ICs are considered. For the first year, the TTL version is the more expensive implementation, and the EPLD numbers look very favorable.

The "True mfg. cost/system" line results after additional costs are figured in. Here we see that the first year, the EPLD version already provides a $1 savings over the PAL version, and that the cost of the TTL implementation is very high. Also, the inserted cost per IC at this point is, $1.15 for TTL, $2.40 for PAL and $1.80 for the EPLD. This is in line with the inserted costs that we mentioned earlier.

The production costs for two additional years shows that the decreasing price of EPLDs (based on the curve of Figure 5) will continue to provide costs savings as production ramps up in quantities.

In terms of functional benefits, the EPLD implementation is the most beneficial because;

- The chip count has gone down, one EPLD has replaced 11 TTL ICs in one implementation, and 2 PALS in the other, reducing the cost and time of:
  —board layout
  —board fab
  —assembly
  —rework
- The reliability of the board has increased. Fewer components translates into less probability of error.
- Modifications are easier to make. Instead of cuts and jumpers (for TTL), or throwing away a PAL, a change is re-programmed.
- The need for de-coupling caps is reduced. All those individual ICs are eliminated and in some cases the distributed capacitance of the board may be enough de-coupling.
- Power supply requirements are small. The active current requirements are much smaller with EPLDs. This in turn reduces the need for large power supplies and fans.
- Cable requirements and enclosure benefits have been improved. Since EPLDs provide better integration over TTL and PALs, the size of the system will be smaller. This translates into fewer boards and cables.
- Inventory is reduced. One EPLD replaces many TTL devices. Also, "on the shelf" programmed EPLDs can be reused in a pinch, PALs can't.

Less expense and probability of "escapes". The time and cost of finding and fixing escape problems is reduced to one reprogrammable IC. In the field, this translates into less "down time" for the customer and a higher level of customer "goodwill" for the OEM.

Allows capability for customized hardware. Specific customer requirements can be implemented. Also, DIP switches and configuration jumpers may not be necessary in many cases, since configurations can be programmed into the EPLD.

Development Costs

As mentioned earlier, the costs of development are usually dismissed as NRE. One reason for this is the difficulty in pegging down these costs. However, while money might be expendable at this stage, time is usually critical. Time saved at the front end can make a difference in beating the competition to market. The following topics are presented for consideration. No costs are assigned to them.

RESEARCH

The amount of time spent researching components, component sources, and technical data can be very large. Designs done with a large IC count require more research and analysis time. Higher integration devices require learning curve time, but, in the long run this tends to reduce research time, especially in future designs.

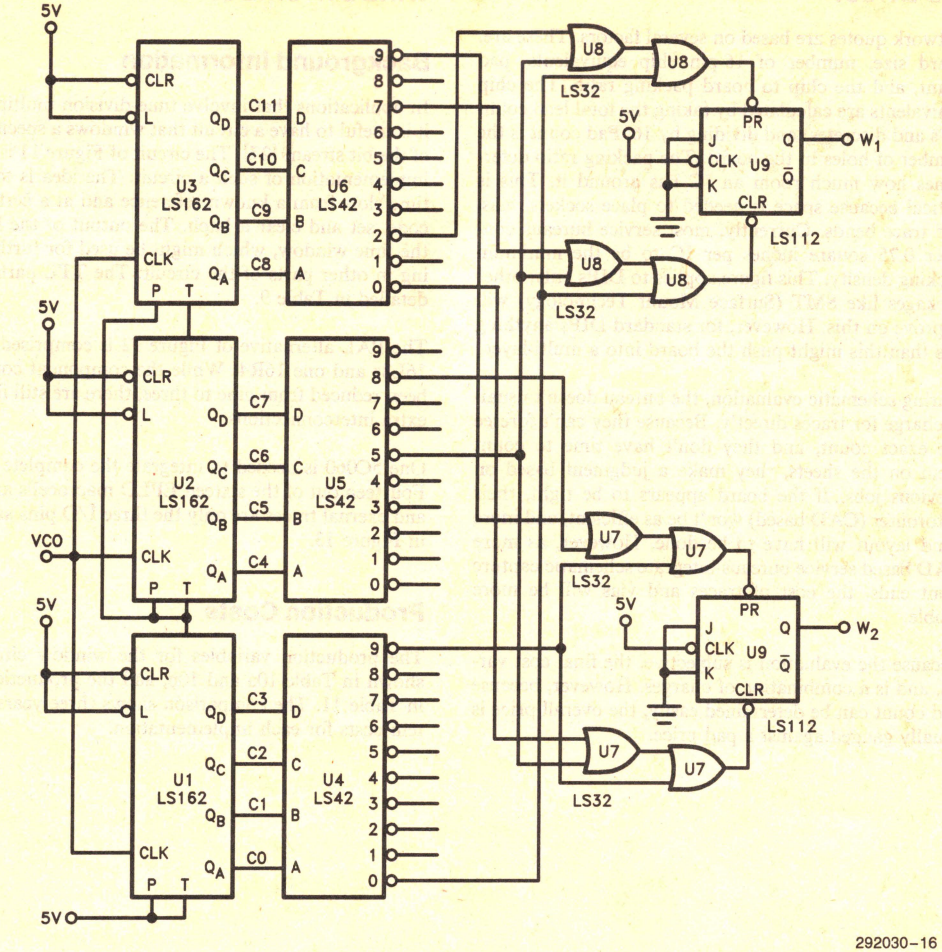
PROTOTYPING

For most companies, prototypes are three to five level wire wrap boards built by inhouse technicians or outside contractors. During prototype fab, a certain amount of work has to be done to each IC. Part of this work is, adding bypass caps, labeling chips, and lead forming. In smaller companies, the board might be hand wrapped. Larger companies might use an automatic wrapper. Once the board is wrapped, a continuity check is done on each wire net to insure connections and minimize shorts.

The turn around time for a protoboard is one to two weeks and can be shortened by paying a premium price. An alternate way of shortening this time is to simplify the board by using denser ICs.

DEBUGGING

Fixing bugs on a protoboard involves unwrapping and wrapping connections, as well as replacing ICs. Making mods on a TTL board is very time consuming and error prone due to the large numbers of wires. Making mods with PALs is expensive since the part usually has to be junked. EPLDs in contrast, are re-programmable and lend themselves to all the revisions that are common in the early design stages.



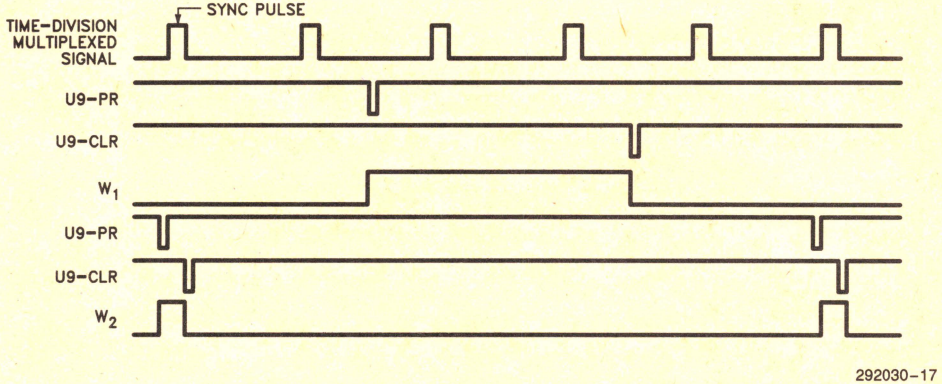


Figure 11. Time Window Generator, TTL Circuit

PCB LAYOUT

Artwork quotes are based on several factors. These are, board size, number of 16-pin chip equivalents, pad count, and the chip to board packing ratio. The chip equivalents are calculated by taking the total lead count (ICs and discretes) and dividing by 16. Pad count is the number of holes in the board. The packing ratio determines how much room an IC has around it. This is critical because space is needed to place sockets, vias, and trace bends. Currently, most service bureaus consider 0.75 square inches per IC to be the minimum packing density. This figure applies to DIPs only, other packages like SMT (Surface Mount Technology) will improve on this. However, for standard DIPs anything less than this might push the board into a multi-layer.

During schematic evaluation, the bureau doesn't usually charge for traces directly. Because they can't foresee the exact count, and they don't have time to count them on the sheets, they make a judgment based on previous jobs. If the board appears to be tight, their autorouter (CAD based) won't be as efficient, and more hand layout will have to be done. However, as more CAD based service bureaus integrate schematic capture front ends, the cost of traces and vias will be more visible.

Because the evaluation is subjective, the final cost varies, and is a combination of charges. However, because pad count can be determined easily, the overall price is usually gauged against a pad price.

WINDOW CIRCUIT

Background Information

In applications that involve time-division multiplexing, it is useful to have a circuit that windows a specific area of the bit stream [27]. The circuit of Figure 11 is a TTL implementation of such a circuit. The idea is to count time slots from a known reference and at a certain decode, set and clear a latch. The output of the latch is the time window, which might be used for further gating in other parts of the circuit. The TTL parts list is detailed in Table 9.

The PAL alternative of Figure 12 is comprised of two 16L8s and one 16R4. While the component count has been reduced from nine to three, there are still fourteen extra interconnections.

One 5C060 is needed to integrate the complete circuit. Fourteen out of the sixteen EPLD macrocells are used, and exernal traces are only the three I/O pins as shown in Figure 13.

Production Costs

The production variables for the window circuit are shown in Table 10a and 10b, and the production costs in Table 11. The comparison shows three years of system costs for each implementation.

Table 9. TTL Component List for Window Generator

IC	Type	DIP	I_{CC}(mA)	Area(In^2)	$
U1	LS162	16	32	.24	.49
U2	LS162	16	32	.24	.49
U3	LS162	16	32	.24	.49
U4	LS42	16	13	.24	.39
U5	LS42	16	13	.24	.39
U6	LS42	14	13	.24	.39
U7	LS32	14	9.8	.21	.18
U8	LS32	14	9.8	.21	.18
U9	LS112	14	6	.21	.29

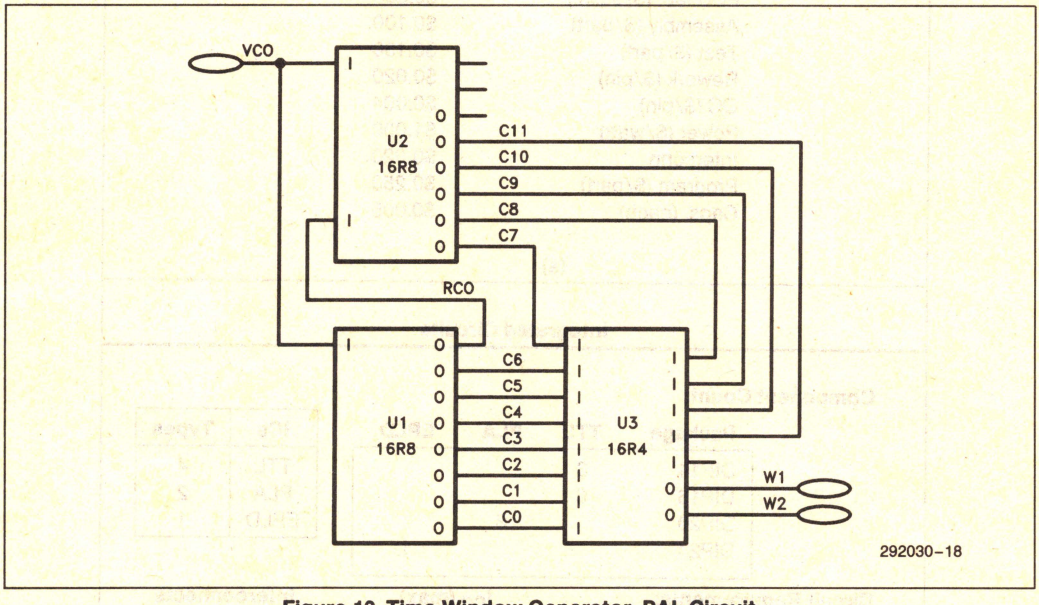


Figure 12. Time Window Generator, PAL Circuit

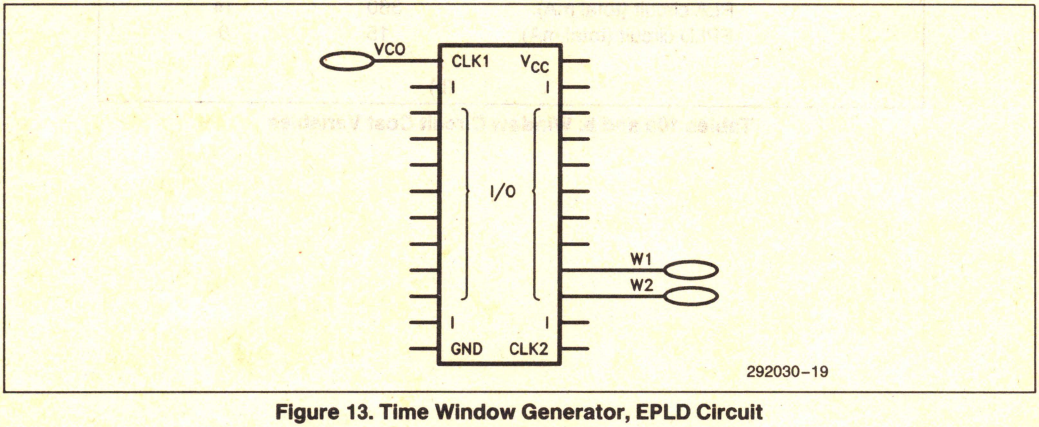


Figure 13. Time Window Generator, EPLD Circuit

Inventory:	Costs
Incoming insp. ($/pin)	$0.010
Storage ($/sq.ft./yr)	$20.000
Maintenance ($/part)	$0.030
Processing ($/part type/yr)	$0.520
Safety stock (%)	2%

Manufacturing:	Costs
PCB fab. ($/sq.in.)	$0.200
Assembly ($/part)	$0.100
Test ($/part)	$0.150
Rework ($/pin)	$0.020
QC ($/pin)	$0.004
Power ($/watt)	$1.000
Interconn	$0.020
Program ($/part)	$0.250
Caps. (each)	$0.005

(a)

Integrated Circuits

Component Count:

Package	TTL	PLA	EPLD
DIP14	3		
DIP16	6		
DIP20		3	
DIP24			1

ICs	Types
TTL	4
PLA	2
EPLD	1

Circuit Requirements:	I_{CC} (max)	Interconnects
TTL circuit (total mA).	160	52
PLA circuit (total mA).	360	14
EPLD circuit (total mA).	15	0

(b)

Tables 10a and b. Window Circuit Cost Variables

Table 11. Window Circuit Production Costs

AVERAGE COMPONENT COST									
	Year 1			Year 2			Year 3		
Package	**TTL**	**PLA**	**EPLD**	**TTL**	**PLA**	**EPLD**	**TTL**	**PLA**	**EPLD**
DIP14	$0.22			$0.19			$0.17		
DIP16	$0.44			$0.37			$0.26		
DIP20		$2.00			$1.70			$1.56	
DIP24			$6.00			$4.20			$2.90
PRODUCTION COSTS									
	Year 1			Year 2			Year 3		
Item (costs per part)	**TTL**	**PLA**	**EPLD**	**TTL**	**PLA**	**EPLD**	**TTL**	**PLA**	**EPLD**
Components	$0.367	$2.000	$6.000	$0.310	$1.700	$4.200	$0.230	$1.560	$2.900
Incoming Insp.	$0.153	$0.200	$0.240	$0.153	$0.200	$0.240	$0.153	$0.200	$0.240
Inventory									
Maintenance	$0.029	$0.038	$0.045	$0.029	$0.038	$0.045	$0.029	$0.038	$0.045
Storage	$0.032	$0.042	$0.050	$0.032	$0.042	$0.050	$0.032	$0.042	$0.050
Processing	$0.231	$0.347	$0.520	$0.231	$0.347	$0.520	$0.231	$0.347	$0.520
Printed Circuit Board									
Fabrication	$0.046	$0.060	$0.072	$0.046	$0.060	$0.072	$0.046	$0.060	$0.072
Trace costs	$0.116	$0.093	$0.000	$0.116	$0.093	$0.000	$0.116	$0.093	$0.000
Assembly	$0.096	$0.125	$0.150	$0.096	$0.125	$0.150	$0.096	$0.125	$0.150
Board test	$0.150	$0.150	$0.150	$0.150	$0.150	$0.150	$0.150	$0.150	$0.150
Rework	$0.015	$0.020	$0.024	$0.015	$0.020	$0.024	$0.015	$0.020	$0.024
QC	$0.061	$0.080	$0.096	$0.061	$0.080	$0.096	$0.061	$0.080	$0.096
Power Supply	$0.089	$0.600	$0.075	$0.089	$0.600	$0.075	$0.089	$0.600	$0.075
Total Cost/Part	$1.385	$3.754	$7.422	$1.328	$3.454	$5.622	$1.248	$3.314	$4.322
Total Cost/System	$12.463	$11.263	$7.422	$11.953	$10.363	$5.622	$11.233	$9.943	$4.322
Additional Costs/System									
Programming loss	$0.000	$0.120	$0.000	$0.000	$0.102	$0.000	$0.000	$0.094	$0.000
Safety stock	$0.165	$0.600	$0.120	$0.140	$0.510	$0.084	$0.104	$0.468	$0.058
Programming fee	$0.000	$0.750	$0.250	$0.000	$0.750	$0.250	$0.000	$0.750	$0.250
De-coupling caps	$0.045	$0.015	$0.005	$0.045	$0.015	$0.005	$0.045	$0.015	$0.005
True mfg. cost/system	$12.673	$12.748	$7.797	$12.137	$11.740	$5.961	$11.381	$11.269	$4.635

The production costs again show that the system cost for the first year is better with EPLDs. The two consecutive years show that the declining price of EPLDs make them an excellent candidate for systems that will ramp up production at that time.

Window Circuit Conclusion

The TTL version of the circuit was implemented with MSI counters and decoders. As a result, the PAL implementation was bound by the number of count bits and had to be programmed into two PALs. In circuits like this, it is useful to rewire the decode for different counts depending on the application. The PAL implementation allows this by incorporating the decode and output latches into one IC.

The EPLD implementation tackles the MSI integration quite easily and also provides the capability to reprogram the decoder. Since the counter and output latches consist of fourteen registered outputs, the sixteen macrocells of the 5C060 easily accommodate the needed functions.

SUMMARY

We have examined the hidden costs of production and how they differ for several logic alternatives. By examining these costs, we have shown that while an EPLD is presently a more expensive part, it's level of integration reduces system costs and improves reliability. The following items should be considered when evaluating logic alternatives:

- system cost is determined by more than component cost
- system cost and reliability is influenced by the type and amount of components used
- semiconductors have a life cycle that determines their present price at design, and at production time

In summary, when all system costs are considered, EPLDs can provide cost savings to the design and production of most board designs.

REFERENCES

1. The Future Is Now: Extending CAE into test of custom VLSI.
Robert S. Broughton, Tektronix.
Michael G. Brashler, Tektronix.
IEEE International Test Conference Proceedings, 1984

2. Reducing The Cost of Quality Through Test Data Managment.
Paul N. Manikas, GenRad Inc.
Stephen G. Eichenlaub, Harvard University.
IEEE International Test Conference Proceedings, 1983

3. A Quantitative Analysis Of The Trade-offs Between Higher Capital Investment and Higher Yield In PCB Testing.
Mark A. Myers, Teradyne Inc.
IEEE International Test Conference Proceedings, 1984

4. An Analysis Of The Cost And Quality Impact Of LSI/VLSI Technology On PCB Test Strategies.
Mark A. Myers, Teradyne Inc.
IEEE International Test Conference Proceedings, 1983

5. IC Quality Control By The User.
Roger Dunn, Xerox Corp.
IEEE International Test Conference Proceedings, 1983

6. An Analysis Of The Economics of Self Test.
P. Varma, University of Manchester.
A. P. Ambler, University of Manchester.
K. Baker, GEC Research Labs.
IEEE International Test Conference Proceedings, 1984

7. In Circuit Testability Factors: Shoot With A Rifle.
Douglas W. Raymond, Zehntel Production Services.
IEEE International Test Conference Proceedings, 1984

8. Seven Steps To Zero Defects.
D. W. Rudd, AT&T Technologies.
Circuits Manufacturing, June 1986

9. Rework Forum
Donald Ford, Senior Editor.
Circuits Manufacturing, September 1986

10. Manufacturing Defect Analyzers: Annual Roundup.
Evaluation Engineering magazine, August 1986

11. Assembly: Automation Makes It Better.
Roland W. Roy and Gordon Weeks, Andover Controls.
Circuits Manufacturing, February 1986

12. Shrinking Lines Squeeze Processes.
Jerry Murray, West Coast Editor.
Circuits Manufacturing, September 1986

13. Ribbon Cable for Reliable Interconnections.
Bennett W. Brachman, Xport Trading Inc.
Electronic Packaging and Production magazine, July 1986

14. TQC and JIT: Partners In Production.
Rick Walleigh, Hewlett Packard.
Circuits Manufacturing, February 1986

15. Automated Handling/Sorting: Multisite Development Moves to Back Burner.
Evaluation Engineering magazine, May 1986

16. Software Charts The Course of Component Testing.
Ronald Pound, Editor.
Electronic Packaging and Production magazine, June 1986

17. Complexity, PLDs Drive The Market.
Evaluation Engineering magazine, July 1986

18. Intel User Defined Logic Handbook.
Intel Corp. 1986

19. VLSI Semicustom Design Guide.
CMP Publications, Summer 1986

20. AMD Programmable Array Logic Handbook.
Advanced Micro Devices, 1984

21. Handbook Of Industrial Engineering.
Gavriel Salvendy, Editor, Purdue University
John Wiley & Sons Publications

22. Components Quality/Reliability Handbook.
Intel Corporation.

23. The Cost Edge.
DM DATA Corp.
Scottsdale, AZ

24. Semiconductor Purchasing Strategies Integrated Circuits Engineering Corp.
Scottsdale, AZ

25. Status 1986 Integrated Circuits Engineering Corp.
Scottsdale, AZ

26. EDN Semicustom Design Series
EDN Magazine, 1985

27. EDN Design Ideas
EDN Magazine, 1985

Techniques for Modular EPLD Designs

Lawrence Palley
PLDO Product Marketing Manager
Intel Corporation
151 Blue Ravine Road
Folsom, CA 95630

INTRODUCTION

Advances in both programmable logic devices and the tools used to configure them now enable new design techniques for custom logic applications. New high capacity flexible architectured EPLDs (erasable and electrically programmable logic devices) allow for complete single chip integration of one or more logic configurations. Additionally, development tools make use of these capabilities by providing alternatives for design input, high speed logic compilation and minimization, heuristic logic fitting into EPLD devices, and superior reporting documentation. Designers can take advantage of these advances with a new Modular EPLD logic design (MELD)technique, to accelerate their product development.

ADVANCES IN EPLDs

Traditional PLDs relied on boolean equation entry and compilation methods for combinatorial function implementation. The primary applications were as SSI/MSI replacements for implementing decode "glue" in microprocessor based systems. PLDs came in bipolar versions with total logic content under 400 gates of equivalent logic. Tools to develop the programmable logic implementation of a function didn't require a high degree of sophistication - the devices for which they were optimizing designs had relatively little logic and little logic flexibility.

Newer EPLDs incorporate several features which broaden their application base. Besides their low power CMOS technology, they incorporate individually configurable register and I/O logic for each macrocell. Devices such as the 5C060 incorporate 16 macrocells with registers programmable into D, or JK configurations. Each register is also programmably configured to be clocked by synchronous or asynchronous clocks. Additionally, outputs and feedback paths for each pin can be combinatorial or registered. The combination of this level of flexibility and gate counts of some devices exceeding 1200, EPLDs have moved programmable logic well past simple combinatorial functions.

To make optimum use of the new EPLD device technology, design tools needed to improve to allow more freedom of design input, better logic optimization for maximum device utilization, and improved reporting documentation. Intel's programmable logic development system provides these improvements. Input methods include the choice of (and combination of) schematic, netlist, state machine, or boolean entry. Besides boolean equation minimization,the optimizer program optimally matches I/O and register resources required by the design with what's available in EPLD devices. It then reports on how the logic entry was reduced, which resources were re- quired, and how the design was placed in a given device. Resources still available in the devices or not able to fit in to the device are also documented.

MELD

Making use of both the advances in EPLD devices and their development tools, engineers can now design hardware (logic) in much the same way as software is developed. This new design technique called MELD (Modular EPLD Logic Design) is shown in Figure 1. Design Entry, in any of the typical engineering formats, is entered on a development station (in this case a personal computer). Using EPLD development system software, the design is then compiled for EPLD implementation. Object code or (in the case of an EPLD) a JEDEC 1's and 0's file are the result. The unique capability of EPLDs is to test a part of a partitioned design in silicon, erase the EPLD, test the next design, and finally to merge the designs together. This powerful logic design methodology allows for the partitioning of a complex logic function into smaller sub-functions that can be individually designed and debugged using the design tools and the erasability feature of EPLDs. After the individual modules are proved to be functional as desired, they can be combined on the same EPLD, allowing for higher integration and its attendant benefits.

"This manuscript originally prepared for and presented at Electro/86."

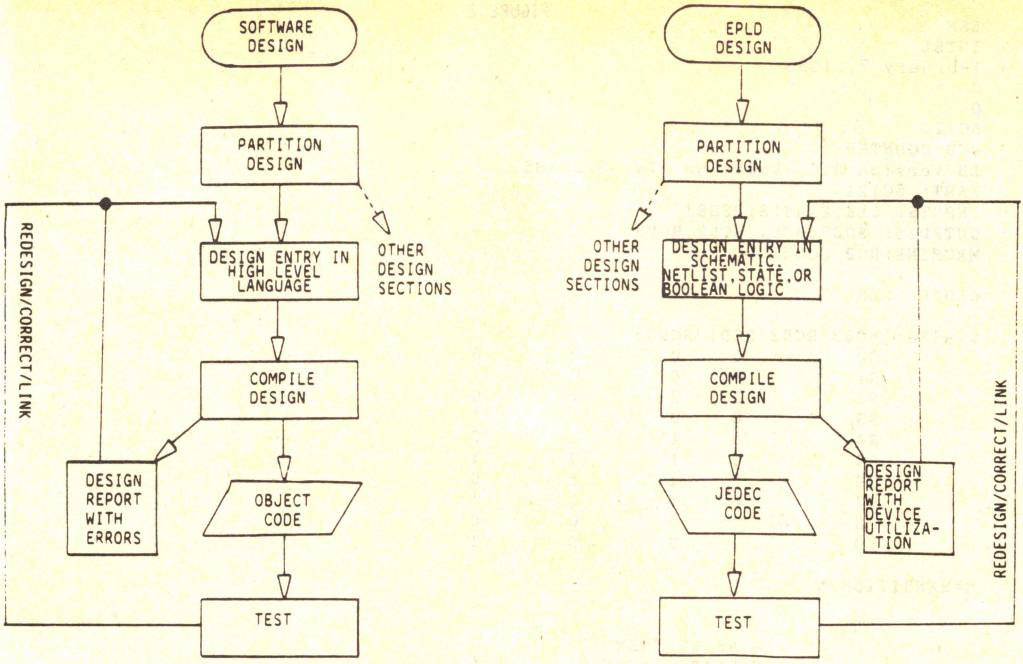


Figure 1. EPLD design process compared with software design process

NOTE: The MELD technique would involve this design process shown above to be implemented for different sub-modules and combining of the sub-functions into the completed high-integration EPLD design.

A MELD Example:

This Modular EPLD logic design (MELD) methodology is now illustrated with an example. The example shown here is a design which implements a BCD-counter which is encoded into a seven-segment display.

Figure 2 shows the design of a BCD-counter designed using state machine entry. This design was compiled (Figure 3) and individually tested in-circuit.

Figure 4 shows a design for implementing the seven-segment display shown in Figure 5. It uses boolean design methods, although not yet optimized. This design has been tested out in several previous designs. An additional section called "LINK EQUATIONS" is now used to connect the BCD-counter with the seven-segment display.

The two design files, BCD-Counter and SEGEQS, are now compiled together in the LOC (Logic Optimizing Compiler) of the Intel Programmable Logic Development System (Figure 6) to yield the combined file, BCD-Counter, of Figure 7.

When implemented in the 5C121 EPLD, the utilization report of Figure 8 results. It shows a pinout designated by the compiler, the routing of inputs, the source of outputs, unused device resources, and some figure of merit about how the design used 5C121 resources. This data can be used to test the device to as feedback for new design inputs. An example of such an input would be to assign signals to 5C121 pins so that PCB layout is simple.

FIGURE 2

```
LSP
INTEL
February 7, 1986
0
0
5C121
BCD COUNTER
LB Version 3.0, Baseline 17x, 9/26/85
PART: 5C121
INPUTS: CLK,ENABLE,RESET
OUTPUTS: BCD0,BCD1,BCD2,BCD3
MACHINE:BCD_COUNTER

CLOCK: CLK

STATES:[BCD3 BCD2 BCD1 BCD0]
        S0      [  0    0    0    0  ]
        S1      [  0    0    0    1  ]
        S2      [  0    0    1    0  ]
        S3      [  0    0    1    1  ]
        S4      [  0    1    0    0  ]
        S5      [  0    1    0    1  ]
        S6      [  0    1    1    0  ]
        S7      [  0    1    1    1  ]
        S8      [  1    0    0    0  ]
        S9      [  1    0    0    1  ]

%TRANSITIONS%

        S0:
                IF RESET THEN S0
                IF ENABLE THEN S1
        S1:
                IF RESET THEN S0
                IF ENABLE THEN S2
        S2:
                IF RESET THEN S0
                IF ENABLE THEN S3
        S3:
                IF RESET THEN S0
                IF ENABLE THEN S4
        S4:
                IF RESET THEN S0
                IF ENABLE THEN S5
        S5:
                IF RESET THEN S0
                IF ENABLE THEN S6
        S6:
                IF RESET THEN S0
                IF ENABLE THEN S7
        S7:
                IF RESET THEN S0
                IF ENABLE THEN S8
        S8:
                IF RESET THEN S0
                IF ENABLE THEN S9
        S9:
                IF RESET THEN S0
                IF ENABLE THEN S0

END$
```

FIGURE 3

```
LSP
INTEL
February 7, 1986
0
0
5C121
BCD COUNTER
LB Version 3.0, Baseline 17x, 9/26/85SMV Version 1.0 Baseline 1.3 85/12/13 00:12:5

PART: 5C121

INPUTS:
CLK, ENABLE, RESET

OUTPUTS:
BCD0, BCD1, BCD2, BCD3

NETWORK:
CLK = INP(CLK)
ENABLE = INP(ENABLE)
RESET = INP(RESET)
%
I/O's for State Machine "BCD_COUNTER"
%
BCD3, BCD3 = RORF(BCD3.d, CLK, GND, GND, VCC)
BCD2, BCD2 = RORF(BCD2.d, CLK, GND, GND, VCC)
BCD1, BCD1 = RORF(BCD1.d, CLK, GND, GND, VCC)
BCD0, BCD0 = RORF(BCD0.d, CLK, GND, GND, VCC)

EQUATIONS:
%
Boolean Equations for State Machine "BCD_COUNTER"
%
%
Current State Equations for "BCD_COUNTER"
%
S0 = BCD3'*BCD2'*BCD1'*BCD0';
S1 = BCD3'*BCD2'*BCD1'*BCD0;
S2 = BCD3'*BCD2'*BCD1*BCD0';
S3 = BCD3'*BCD2'*BCD1*BCD0;
S4 = BCD3'*BCD2*BCD1'*BCD0';
S5 = BCD3'*BCD2*BCD1'*BCD0;
S6 = BCD3'*BCD2*BCD1*BCD0';
S7 = BCD3'*BCD2*BCD1*BCD0;
S8 = BCD3*BCD2'*BCD1'*BCD0';
S9 = BCD3*BCD2'*BCD1'*BCD0;
%
SV Defining Equations for State Machine "BCD_COUNTER"
%
BCD3.d = S8.n
       + S9.n;
BCD2.d = S4.n
       + S5.n
       + S6.n
       + S7.n;
BCD1.d = S2.n
       + S3.n
       + S6.n
       + S7.n;
BCD0.d = S1.n
       + S3.n
       + S5.n
       + S7.n
       + S9.n;
%
```

FIGURE 3 (CONTINUED)

```
Next State Equations for State Machine "BCD COUNTER"
%
S1.n = S1 * (ENABLE)' *  RESET)'
     + S0 * ENABLE * (RESET)':
S2.n = S2 * (ENABLE)' *  RESET)'
     + S1 * ENABLE * (RESET)':
S3.n = S3 * (ENABLE)' * (RESET)'
     + S2 * ENABLE * (RESET)':
S4.n = S4 * (ENABLE)' * (RESET)'
     + S3 * ENABLE * (RESET)':
S5.n = S5 * (ENABLE)' * (RESET)'
     + S4 * ENABLE * (RESET)':
S6.n = S6 * (ENABLE)' *  RESET)'
     + S5 * ENABLE * (RESET)':
S7.n = S7 * (ENABLE)' * (RESET)'
     + S6 * ENABLE * (RESET)':
S8.n = S8 * (ENABLE)' * (RESET)'
     + S7 * ENABLE * (RESET)':
S9.n = S9 * (ENABLE)' * (RESET)'
     + S8 * ENABLE * (RESET)':

END$
```

FIGURE 4

```
LSP
INTEL
February 7, 1986
0
0
5C121
SEVEN SEGMENT DECODERS FOR BCD COUNTER
LB Version 3.0, Baseline 17x, 9/26/85
PART: 5C121
INPUTS:
OUTPUTS: SEGA,SEGB,SEGC,SEGD,SEGE,SEGF,SEGG
NETWORK:
SEGA = CONF (SEGA,VCC)
SEGB = CONF (SEGB,VCC)
SEGC = CONF (SEGC,VCC)
SEGD = CONF (SEGD,VCC)
SEGE = CONF (SEGE,VCC)
SEGF = CONF (SEGF,VCC)
SEGG = CONF (SEGG,VCC)
EQUATIONS:
SEGA = 0 + 2 + 3 + 5 + 7 + 8 + 9;

SEGB = 0 + 1 + 2 + 3 + 4 + 7 + 8 + 9;

SEGC = 0 + 1 + 3 + 4 + 5 + 6 + 7 + 8 + 9;

SEGD = 0 + 2 + 3 + 5 + 6 + 8;
SEGE = 0 + 2 + 6 + 8;
SEGF = 0 + 4 + 5 + 6 + 8 + 9;
SEGG = 2 + 3 + 4 + 5 + 6 + 8 + 9;

0 = /D3*/D2*/D1*/D0;
1 = /D3*/D2*/D1*D0;
2 = /D3*/D2* D1*/D0;
3 = /D3*/D2* D1*D0;
4 = /D3* D2*/D1*/D0;
5 = /D3* D2*/D1*D0;
6 = /D3* D2* D1*/D0;
7 = /D3* D2* D1*D0;
8 = D3*/D2*/D1*/D0;
9 = D3*/D2*/D1*D0;
%LINK EQUATIONS %

D0 = BCD0;
D1 = BCD1;
D2 = BCD2;
D3 = BCD3;

END$
```

FIGURE 6
Intel Programmable Logic Software

```
LOC Menu
F1 Help
F2 iPLS Menu
F3 Input Format        ADF
F4 File Name           A:BCD A:SEGEQS
F5 Minimization        Yes
F6 Inversion Control   No
F7 LEF Analysis        Yes
```

FIGURE 7

```
LSP
INTEL
February 7, 1986
0
0
5C121
BCD COUNTER
LB Version 3.0, Baseline 17x, 9/26/85SMV Version 1.0 Baseline 1.3 85/12/13 00:12:5

PART:
        5C121
INPUTS:
        CLK, ENABLE, RESET
OUTPUTS:
        BCD0, BCD1, BCD2, BCD3, SEGA, SEGB, SEGC, SEGD, SEGE, SEGF, SEGG
NETWORK:
        CLK = INP(CLK)
        ENABLE = INP(ENABLE)
        RESET = INP(RESET)
        BCD0, BCD0 = RORF(BCD0.d, CLK, GND, GND, VCC)
        BCD1, BCD1 = RORF(BCD1.d, CLK, GND, GND, VCC)
        BCD2, BCD2 = RORF(BCD2.d, CLK, GND, GND, VCC)
        BCD3, BCD3 = RORF(BCD3.d, CLK, GND, GND, VCC)
        SEGA = CONF(SEGA, VCC)
        SEGB = CONF(SEGB, VCC)
        SEGC = CONF(SEGC, VCC)
        SEGD = CONF(SEGD, VCC)
        SEGE = CONF(SEGE, VCC)
        SEGF = CONF(SEGF, VCC)
        SEGG = CONF(SEGG, VCC)
EQUATIONS:
        SEGG = BCD1 * BCD3' * BCD2'
             + BCD1' * BCD3' * BCD2
             + BCD1' * BCD3 * BCD2'
             + BCD1 * BCD3' * BCD0';

        SEGF = BCD3' * BCD1' * BCD0'
             + BCD3' * BCD2 * BCD1'
             + BCD3 * BCD2' * BCD1'
             + BCD3' * BCD2 * BCD0';

        SEGE = BCD2' * BCD1' * BCD0'
             + BCD3' * BCD1 * BCD0';

        SEGD = BCD2' * BCD1' * BCD0'
             + BCD3' * BCD2' * BCD1
             + BCD3' * BCD1 * BCD0'
             + BCD3' * BCD2 * BCD1' * BCD0;

        SEGC = BCD2' * BCD1'
             + BCD3' * BCD2
             + BCD3' * BCD0;

        SEGB = BCD2' * BCD1'
             + BCD3' * BCD0'
             + BCD3' * BCD1;

        SEGA = BCD3' * BCD2' * BCD0'
             + BCD3 * BCD2' * BCD1'
             + BCD3' * BCD2' * BCD1
             + BCD3' * BCD2 * BCD0;

        BCD3.d = BCD3 * BCD2' * BCD1' * BCD0' * RESET'
               + BCD3 * BCD2' * BCD1' * ENABLE' * RESET'
               + BCD3' * BCD2 * BCD1 * BCD0 * ENABLE * RESET';
```

FIGURE 7 (CONTINUED)

```
        BCD2.d = BCD3' * BCD2 * BCD0' * RESET'
               + BCD3' * BCD2 * BCD1' * RESET'
               + BCD3' * BCD2 * ENABLE' * RESET'
               + BCD3' * BCD2' * BCD1 * BCD0 * ENABLE * RESET';

        BCD1.d = BCD3' * BCD1 * BCD0' * RESET'
               + BCD3' * BCD1 * ENABLE' * RESET'
               + BCD3' * BCD1' * BCD0 * ENABLE * RESET';

        BCD0.d = BCD3' * BCD0' * ENABLE * RESET'
               + BCD3' * BCD0 * ENABLE' * RESET'
               + BCD2' * BCD1' * BCD0 * ENABLE' * RESET'
               + BCD2' * BCD1' * BCD0' * ENABLE * RESET';

  END$
```

FIGURE 8

```
Logic Optimizing Compiler Utilization Report

 ***** Design implemented successfully

LSP
INTEL
February 7, 1986
0
0
5C121
BCD COUNTER
LB Version 3.0, Baseline 17x, 9/26/85SMV Version 1.0 Baseline 1.3 85/12/13 00:12:5


                5C121
              - - - - -
        CLK - | 1   40 | - Vcc
        GND - | 2   39 | - Vcc
        GND - | 3   38 | - ENABLE
        GND - | 4   37 | - RESET
        GND - | 5   36 | - GND
        GND - | 6   35 | - GND
        GND - | 7   34 | - GND
       SEGD - | 8   33 | - GND
  RESERVED - | 9   32 | - SEGG
  RESERVED - |10   31 | - RESERVED
  RESERVED - |11   30 | - RESERVED
       SEGA - |12   29 | - SEGC
  RESERVED - |13   28 | - SEGB
  RESERVED - |14   27 | - RESERVED
       SEGE - |15   26 | - RESERVED
  RESERVED - |16   25 | - SEGF
       BCD2 - |17   24 | - RESERVED
  RESERVED - |18   23 | - BCD3
  RESERVED - |19   22 | - BCD1
        GND - |20   21 | - BCD0
              - - - - -
```

FIGURE 8 (CONTINUED)

INPUTS

Name	Pin	Resource	MCell #	PTerms	MCells	Feeds: OE	Clear	Clock
CLK	1	INP	-	-	-	-	-	Reg
RESET	37	INP	-	-	10 11 12 19	-	-	-
ENABLE	38	INP	-	-	10 11 12 19	-	-	-

OUTPUTS

Name	Pin	Resource	MCell #	PTerms	MCells	Feeds: OE	Clear
SEGD	8	CONF	28	4 4	-	-	-
SEGA	12	CONF	24	4 6	-	-	-
SEGE	15	CONF	21	2 4	-	-	-
BCD2	17	RORF	19	4 4	1 4 5 8 10 12 19 21 24 28	-	-
BCD0	21	RORF	12	4/ 8	1 4 5 8 10 11 12 19 21 24 28	-	-
BCD1	22	RORF	11	3/ 8	1 4 5 8 10 11 12 19 21 24 28	-	-

FIGURE 8 (CONTINUED)

BCD3	23	RORF	10	3	4	1 4 5 8 10 11 12 19 21 24 28	-	-
SEGF	25	CONF	8	4/	4	-	-	-
SEGB	28	CONF	5	3/	6	-	-	-
SEGC	29	CONF	4	3/	6	-	-	-
SEGG	32	CONF	1	4/	4	-	-	-

UNUSED RESOURCES

Name	Pin	Resource	MCell	PTerms
-	2	-	-	-
-	3	-	-	-
-	4	-	-	-
-	5	-	-	-
-	6	-	-	-
-	7	-	-	-
-	9	-	27	10
-	10	-	26	8
-	11	-	25	6
-	13	-	23	8
-	14	-	22	10
-	16	-	20	12
-	18	-	18	8
-	19	-	17	8
-	24	-	9	12
-	26	-	7	10
-	27	-	6	8
-	30	-	3	8
-	31	-	2	10
-	33	-	-	-
-	34	-	-	-
-	35	-	-	-
-	36	-	-	-
-	NA	-	13	8
-	NA	-	14	8
-	NA	-	15	8
-	NA	-	16	8

PART UTILIZATION

37%	Pins
39%	MacroCells
18%	Pterms

CONCLUSIONS

The complete design took less than an hour to enter, compile, and link with EPLDs. The ability to partition designs, then individually implement those designs in the logic design entry of choice, and finally to link designs together is a new design method only available with advances in programmable logic and their design tools. By taking advantage of these capabilities, designers can bring logic implementations to market faster and with a high degree of integration.

Crosspoint Switch: A PLD Approach

by Jim Donnell, Intel Corp.

Erasable programmable logic devices (EPLDs) combine the gate densities of low-end gate arrays with the short development time and low cost of EPROMs. This merging of technologies produces a device with features suited to a wide range of digital applications. In contrast to the long development times (and higher costs) for gate arrays, EPLDs require minimal frontend design time. In just a few hours, EPLD designs can be developed, modified and verified. Also, core elements from one EPLD design can be incorporated into new designs as quickly as standard software subroutines from one program can be modified and used in other programs.

The design of a digital crosspoint switch using an Intel 5C121 EPLD illustrates these features. *Digital Design* implemented a crosspoint switch in a gate array last year (see *Digital Design*, January through March, 1985). Applications that require a data transfer from one of several inputs to one of several outputs frequently use a digital crosspoint switch. Using the 5C121 EPLD, Intel Corp. (Santa Clara, CA) designed three different configurations of a crosspoint switch.

Offered in a 40-pin package that provides up to 36 inputs or 24 outputs, the 5C121 supports up to 28 macrocells (including four buried registers) and 236 product terms (p-terms). Logic density in the 5C121 is the equivalent of 1,200 usable NAND gates. Maximum power requirements are 100 mA active and 30 mA standby with TTL input levels. With CMOS input levels, a 5C121 requires 50 mA active and 3 mA standby.

Two major parameters determine the complexity and configuration of a digital crosspoint switch: the number of possible switching locations for each bit (inputs and outputs), and the number of bits transferred in one clock pulse (word width). The availability of I/O pins, macrocells and p-terms for a given EPLD device dictates the number of switches that can be designed into a single device.

Configuration 1

The first circuit (**Figure 1**) considered is a digital crosspoint switch with eight inputs and a 3-bit word width. This switch transfers a 3-bit word coming from one of eight sources to a particular output. The number of devices "OR-tied" to each output pin determines the number of outputs. Selecting one of eight data inputs from each of the three channels (A0 to A7, B0 to B7 and C0 to C7), the switch routes that data to a single output (QA, QB and QC). Each output can be OR-tied to more than one

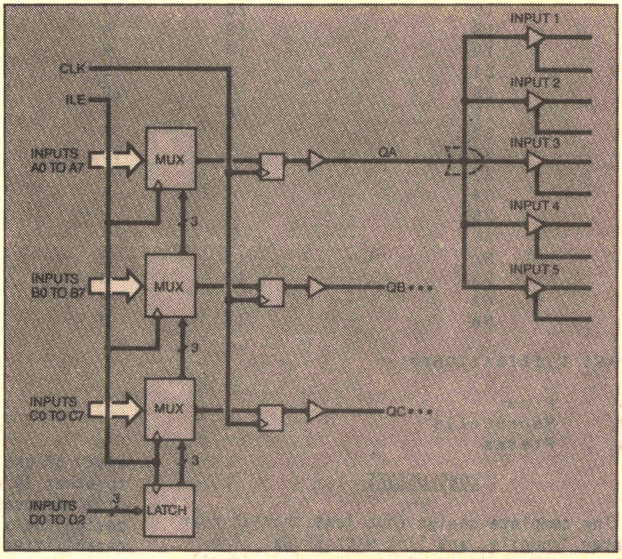


Figure 1: Configuration 1 uses a three-channel eight-to-one multiplexer circuit with latching inputs. Each output can drive multiple, individually selected inputs to complete the digital crosspoint switch. By connecting inputs to the EPLD outputs in an "OR-tied" configuration, with only one input enabled at any time, the multiplexer circuit becomes a crosspoint switch.

© Intel Corporation, 1986
Reprinted with permission from *Digital Design*

three-state input to complete the switch (only one input can be enabled at a time). Three additional control bits (D0 to D2) select one of the eight different inputs. All three channels operate in parallel. Separate input and output clocks allow a high data rate and relax input set-up and hold times. Input data for all three channels, along with the three select bits, are latched by ILE. Data at the inputs can change state after being latched and data is clocked out of the switch by CLK.

Equation 1 shows the Boolean expression for a single channel in the sum-of-products form. (See **Table 1** for all equations.) The Boolean expression for the remaining two channels is similar: the designer need only change the A in the equations to a B or C.

Timing Analysis

The internal delay paths determine the circuit's maximum operating frequency (fmax). In this configuration there is an input delay (Tin), an array delay (Tad), a register delay (Trd) and an output delay (Tod). The fmax is a function of the signals that must settle at the input of the output register before the rising edge of the clock. In this case, signals propagate only through the input latches and the array. Therefore, the data must be valid at the inputs Tin + Tad just nanoseconds before the rising edge of the internal clock signal (CLK). However, because of the inherent delay of the CLK signal, this reference must be shifted to the rising edge of the external clock signal by subtracting the internal clock delay (Tic). The external data set-up time (Tsu) is shown in **Equation 2**. Inverting this time requirement yields the maximum operating frequency.

As the output flip-flops are clocked, data propagates through the register to the output pin. With reference to the external clock pin, data becomes valid at the outputs Tic + Trd + Tod nanoseconds after the rising edge of the clock. **Figure 2** shows the timing requirements for this circuit, including the input latch signal.

Using a 5C121-50 (50-nsec propagation delay), data can be sent through this switch configuration at 25 Mbits/sec. This transfer rate remains independent of the word width. Since one 5C121 EPLD in this configuration can simultaneously transfer three bits of information, three 5C121's are required to transfer a byte of data during each clock cycle. This configuration of a digital crosspoint switch uses 86% of the 40 pins, 71% of the macrocells and 11% of the available p-terms in the 5C121 EPLD.

Configuration 2

The second circuit (**Figure 3**) also selects one of eight inputs (I0 to I7), but this time data is routed to one of eight different

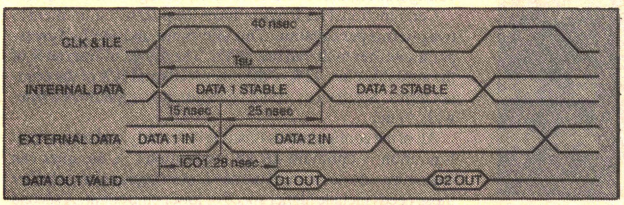


Figure 2: A 40-nsec internal set-up time (prior to clocking data through the output flip-flop) marks Configuration 1. Data clocked into all eight input latches at the rising edge of one ILE/CLK cycle is selected and clocked out of the output flip-flop on the next rising edge of ILE/CLK.

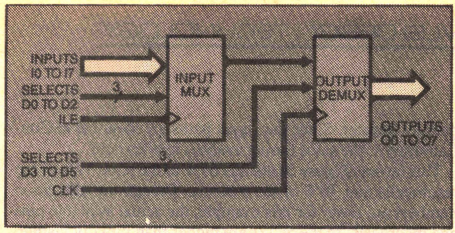


Figure 3: Configuration 2 uses a single-bit eight-input/eight-output digital crosspoint switch. Designers can implement this for either optimal package count (see Figure 4) or for optimal speed (see Figure 5).

outputs (Q0 to Q7). Six control bits are required for each transfer: three to select the input path (D0 to D2); three to select the output path (D3 to D5). By selecting a single output path and clocking all output registers simultaneously, deselected outputs are automatically cleared. This is useful for designs where only the most current data is needed. **Equation 4** is the common equation to select one of eight input paths. **Equations 5 to 12** complete the Boolean equations for this example.

The previous equations would contain eight product terms if they were written in expanded form. However, by treating SELECTEQ as one signal, each equation contains only one product term. Both options are available in the 5C121. But, there

In contrast to the long development times for gate arrays, EPLDs require minimal frontend design time.

are advantages and disadvantages to the two methods. If SELECTEQ is implemented as one signal through a combinational feedback option, one and one-half crosspoint switches can be implemented in one 5C121 (**Figure 4**). The trade-off is faster speed for low chip count. By design, only 18 macrocells in the 5C121 can support eight product terms. On the other hand, selecting the combinational option reduces the p-terms but introduces an additional input mux delay.

Figure 4 shows that an input signal must pass through four delays before reaching the input to the flip-flop. Again, subtracting the input clock delay to shift the reference point yields **Equation 13** for the set-up time. Inverting Tsu gives the maximum operating frequency. In this configuration, data can be clocked through at 12 Mbits/sec. This layout utilizes 97% of the available pins, 89% of the available macrocells and 13% of the product terms. Six 5C121s would be required to implement a byte-wide switch with this layout.

If the combinational feedback option is not used, there are eight output equations, each containing eight product terms. Assigning these equations to the macrocells that support eight p-terms shows that only a single, one-of-eight select line digital crosspoint switch fits into one 5C121. Thus, the design requires eight 5C121s to complete a byte-wide

parallel transfer. Since the signal paths are identical to Configuration 1, the same timing analysis applies here.

This layout (**Figure 5**) utilizes 65% of the pins, 39% of the macrocells and 30% of the p-terms. Though the utilization numbers are lower for this example, the actual available pins and macrocells in the 5C121 are higher than initially visible. Since macrocells in the 5C121 are organized into groups of four, when one output structure in a macrocell group is defined the other three must be of the same structure. Many times, this results in unused pins being labeled "RESERVED" in the utilization report.

Configuration 3

The final circuit (**Figure 6**) again uses eight inputs (I0 to I7) and eight outputs (Q0 to Q7), though this time the deselected outputs "remember" their previously selected state. With the 5C121's register feedback option, deselected outputs can hold the last data bit sent to that output. New data appears when the output is selected again.

Equations 14 to 22 express the Boolean terms necessary to implement this hold feature in the digital crosspoint switch. Note that each output is now a function of both the present inputs and the previous output (Qnfbk), which implements the registered feedback. Data bits D3, D4 and D5 determine which data bit will pass to the output. Again, the number of p-terms dictates the use of combinational feedback, as in Configuration 2.

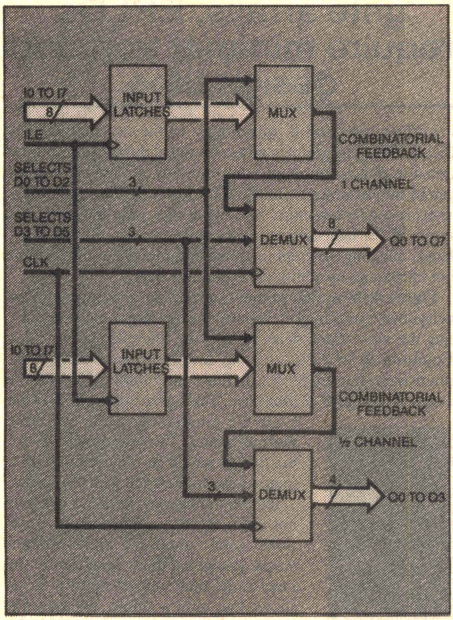


Figure 4: Configuration 2 features a low package count layout. Note that one and one-half switches fit into each 5C121 EPLD. This configuration uses combinatorial feedbacks to simplify the logic equations, thus eliminating the requirement for eight product terms per output.

Table 1. Equations for All Configurations

```
QA = A0•/D2•/D1•/D0 + A1•/D2•/D1•D0                          (1)
   + A2•/D2•D1•/D0 + A3•/D2•D1•D0
   + A4•D2•/D1•/D0 +A5•D2•/D1•D0
   + A6•D2•D1•/D0 + A7•D2•D1•D0;

Tsu = Tin + Tad - Tic                                        (2)

fmax = 1/(Tin + Tad - Tic)                                   (3)

SELECTEQ = I0•/D2•/D1•/D0 + I1•/D2•/D1•D0                    (4)
         + I2•/D2•D1•/D0 + I3•/D2•D1•D0
         + I4•D2•/D1•/D0 + I5•D2•/D1•D0
         + I6•D2•D1•/D0 + I7•D2•D1•D0;

Q0 = /D5•/D4•/D3 • SELECTEQ;                                 (5)
Q1 = /D5•/D4•D3 • SELECTEQ;                                  (6)
Q2 = /D5•D4•/D3 • SELECTEQ;                                  (7)
Q3 = /D5•D4•D3 • SELECTEQ;                                   (8)
Q4 = D5•/D4•/D3 • SELECTEQ;                                  (9)
Q5 = D5•/D4•D3 • SELECTEQ;                                   (10)
Q6 = D5•D4•/D3 • SELECTEQ;                                   (11)
Q7 = D5•D4•D3 • SELECTEQ;                                    (12)

Tsu = Tin + 2(Tad) + Tcf - Tic                               (13)

SELECTEQ = I0•/D2•/D1•/D0 + I1•/D2•/D1•/D0                   (14)
         + I2•/D2•D1•/D0 + I3•/D2•D1•D0
         + I4•D2•/D1•/D0 + I5•D2•/D1•D0
         + I6•D2•D1•/D0 + I7•D2•D1•D0

Q0 = /D5•/D4•/D3•SELECTEQ + /(/D5•/D4•D3)•Q0fbk;             (15)
Q1 = /D5•/D4•D3•SELECTEQ + /(/D5•/D4•D3)•Q1fbk;              (16)
Q2 = /D5•D4•/D3•SELECTEQ + /(/D5•D4•/D3)•Q2fbk;              (17)
Q3 = /D5•D4•/D3•SELECTEQ + /(/D5•D4•D3)•Q3fbk;               (18)
Q4 = D5•/D4•/D3•SELECTEQ + /(D5•/D4•/D3)•Q4fbk;              (19)
Q5 = D5•/D4•D3•SELECTEQ + /(D5•/D4•D3)•Q5fbk;               (20)
Q6 = D5•D4•/D3•SELECTEQ + /(D5•D4•/D3)•Q6fbk;               (21)
Q7 = D5•D4•D3•SELECTEQ + /(D5•D4•D3)•Q7fbk;                (22)
```

Timing Analysis

This configuration's timing analysis is similar to Configuration 2's combinational feedback analysis, with the exception of a register feedback delay (Trf). Trf is the time that the data is present at the output of the flip-flop to the time that data is available to the array.

The total delay associated with the registered feedback consists of the Trd, the Trf and the Tad. Data from the flip-flop output reaches the input in about 50 nsec. The delay associated with data coming from the input pins is the same as that of Configuration 2 with combinational feedback – approximately 83 nsec. Using this as the clock period, there is ample time to implement the register feedback without affecting the cycle time. In this configuration, data could be clocked through at 12 Mbits/sec.

Combinational feedback reduces the p-term requirement to two p-terms per equation. This allows one and one-half crosspoint switches to fit into one 5C121. The design utilizes 64% of the available pins, 42% of the macrocells and 11% of the product terms. Six devices would be required to implement a byte-wide switch.

All of the configurations function differently, and no one configuration is optimum for all applications. A designer can customize a device to meet the needs of an application, whether those needs include higher speed or lower chip count. A second device can be quickly developed for a different application. Designers are no longer restricted to a single device type that must be adapted to an application with additional logic devices.

An original design can be developed in an afternoon. Additional devices derived from an original design can be developed in a few hours. Also, the ability to erase an EPLD and reprogram it allows design errors to be corrected immediately. Instead of several weeks delay with gate arrays, a designer using EPLDs can have working silicon devices in one day.

Both the flexibility and short design times associated with EPLDs make them a good choice for applications that benefit from custom silicon devices. Today, EPLDs offer designers the densities and configuration flexibility of gate arrays, along with the short development time and cost associated with EPROMs.

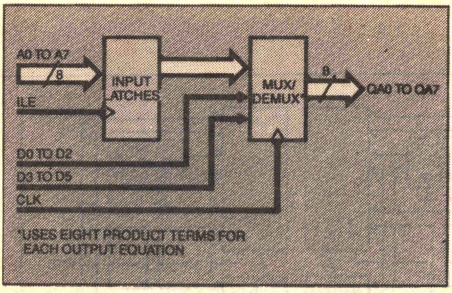


Figure 5: This circuit (Configuration 2 optimized for speed) combines the multiplexer and demultiplexer functions for each channel in a single array. Since each output equation uses eight product terms, only one switching channel can fit into each 5C121 package.

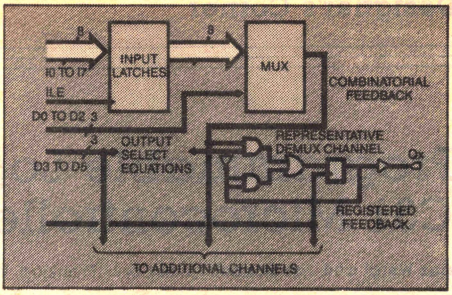


Figure 6: Configuration 3 shows the use of registered feedback to allow deselected outputs to retain their previously selected data. The logic for a representative channel is shown. As with Configuration 2, this configuration can be optimized for package count or speed.

A Programmable Logic Mailbox for 80C31 Microcontrollers

Karlheinz Weigl and Jim Donnell, Intel Corp., Frankfurt, West Germany, and Folsom, CA

This article describes the implementation of a semi-intelligent interface between two 80C31 microcontrollers, using a mailbox protocol. Applications for an interface such as the the one described here are often found in industrial control areas where multiple microcontrollers are used to accomplish a given task. Due to the architecture of the microcontroller (i.e., no READY input; no HOLD/HLDA interface; port-oriented I/O; etc.), exchanging data and status between these devices becomes a cumbersome task. Given this directive, it becomes the designer's task to develop a multi-port memory interface that allows for zero wait-state operation (i.e., no READY signal required), that electrically isolates the microcontroller buses, and that permits asynchronous access. Synchronization would result in the generation of wait states.

We realize the logic necessary to implement the desired functions in two erasable programmable logic devices (EPLDs). One device, the 5C031, contains roughly the equivalent of 300 2-input NAND gates, while the other EPLD, the 5C060, can implement designs with up to approximately 600 gates.

The Mailbox Principle And its Implementation

In a mailbox memory system, the microcontrollers exchange information as bytes of data written to or read from a mailbox register. Control logic permits simultaneous access to the mailbox, thus eliminating the need for arbitration between the microcontrollers. Implementing the data exchange in this form achieves most of the design criteria given above.

Avoiding bus arbitration together with the short propagation delays of the EPLDs provides zero wait-state operation of the data exchange. Electrical isolation of the address and data buses is achieved by using the high-impedance output capability of the 5C060. Simultaneous, asynchronous access is achieved by separating the RD and WR strobes issued by each microcontroller.

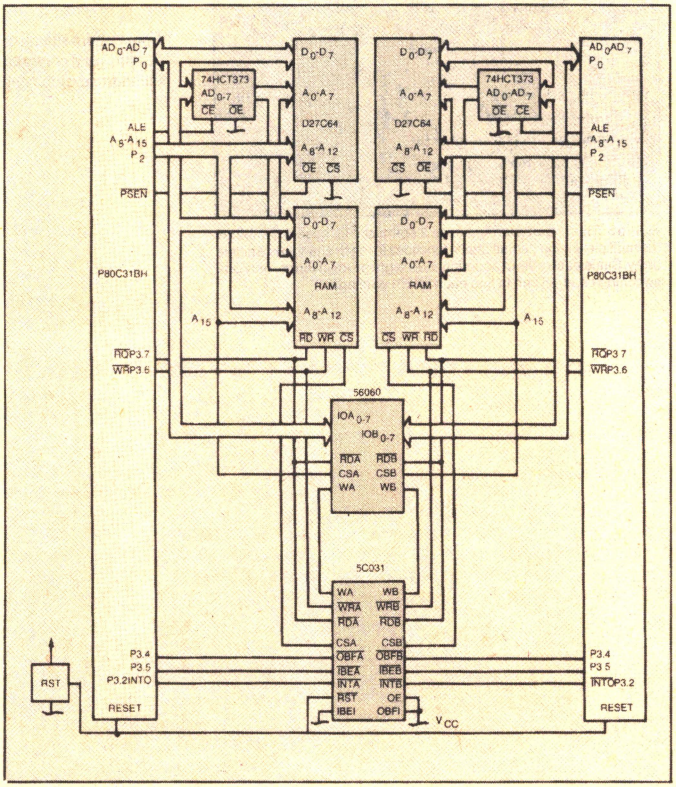


FIGURE 1. Schematic of mailbox memory system.

With a mailbox memory system, there is an obvious need for some type of communication protocol to confirm the reception of a message, or the presence of data in the mailbox. In addition, the read and write logic must be defined such that simultaneous access to the mailbox is permitted. In order to segment the task, the design will be approached in terms of two separate mod-

Copyright© 1987 by CMP Publications, Inc., 600 Community Drive, Manhasset, NY 11030.
Reprinted with permission from VLSI Systems Design.

ules: the mailbox (memory section), and the the control logic (protocol).

To begin the design of the memory section, it is first helpful to identify the resources required for the design. The mailbox requires a total of 16 memory storage registers (two bytes of data), tri-state output control, and two separate clock lines used to write the memory registers.

The 5C060 EPLD was chosen to implement the memory section. This device contains 16 programmable register groups that may be configured to operate as JK-, RS-, D-, and T-type flip-flops. Each register group feeds a bi-directional input/output pin, which may be tri-stated via an output-enable product term. These I/O pins may also serve as data inputs when the register output is tri-stated. This feature forms the basis of the read-signal logic required in the design. Write logic can be accomplished through the two synchronous clock inputs provided in the 5C060. Each synchronous clock drives a set of eight registers in the device. The operation of the memory section of the mailbox memory may now be solidified.

As shown in Figure 1, the two microcontrollers are separated into controller A and controller B. Register group A (signals IOA0 to IOA7) serves as an input buffer to microcontroller A. This buffer receives information from microcontroller B's data bus. The write control for register group A comes from microcontroller B.

Again, referring to Figure 1, it can be seen that register group B serves as an output buffer to microcontroller B. This buffer gets information from microcontroller A and is therefore write-controlled by microcontroller A.

Data Transfer

In order to read data from the mailbox, the microcontroller must initiate a read cycle addressing the mailbox. The read signal (RDA for microcontroller A, RDB for microcontroller B) enables the tri-state outputs of the 5C060, revealing the appropriate data. Spurious read cycles are avoided by logically combining the read signal with a chip select signal (CSA or CSB) within the chip. The example shown utilizes address bit A15 as the

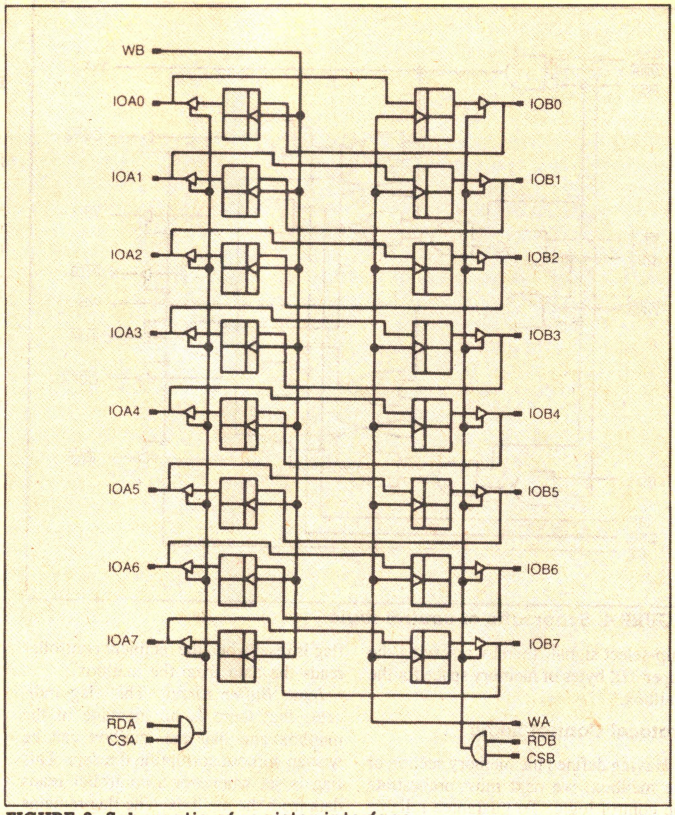


FIGURE 2. Schematic of register interface.

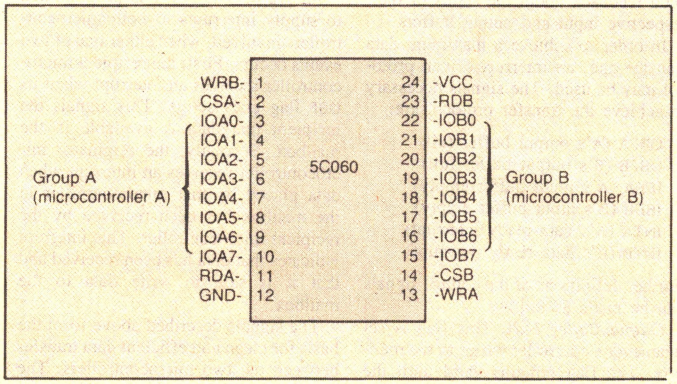


FIGURE 3. Pin-out for register interface.

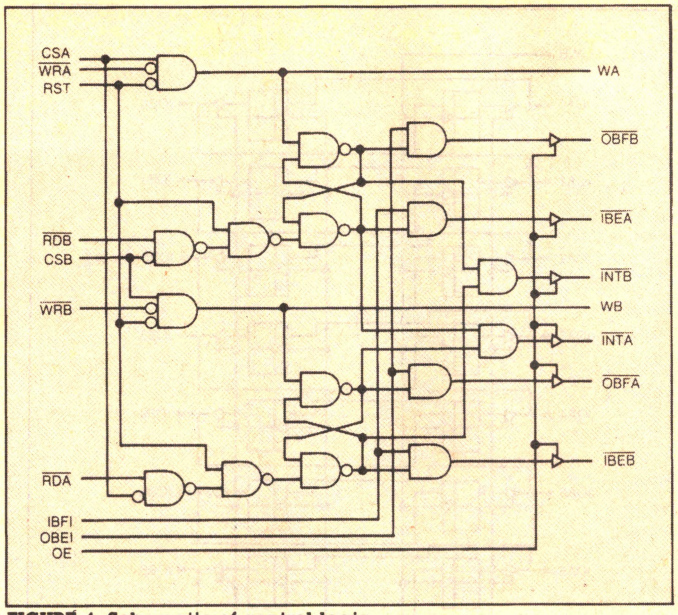


FIGURE 4. Schematic of control logic.

chip-select signal, thereby reserving the upper 32K bytes of memory space for the mailbox.

Protocol Control Logic

Having defined the memory section of the mailbox, we next must orchestrate the control logic. To guarantee reliable data transfers, both microcontrollers need feedback about the status of their respective input and output buffers.

In order to achieve a maximum data transfer rate, an interrupt-driven protocol may be used. The signals necessary to achieve the transfer protocol are:

OBFA (A's output buffer full)
OBFB (B's output buffer full)
IBEA (A's input buffer empty)
IBEB (B's input buffer empty)
INTA (A's data ready interrupt)
INTB (B's data ready interrupt)

Further definitions of the control signals can be made as follows.

Output Buffer Full. This flag is set whenever a controller writes to the mailbox. The flag remains valid until the second controller has read the data. The flag is reset when the recipient controller reads the data from the mailbox.

Input Buffer Empty. This flag indicates that there is no message in the mailbox and that the mailbox can be written without corrupting the data. This flag is set whenever a controller reads data from the mailbox. The flag remains set until data is placed in the mailbox.

Interrupt. The 5C031 is programmed to supply interrupts to both microcontrollers involved, when either one of two events occurs. First, the recipient microcontroller receives an interrupt when its OBF flag goes active. This signals the recipient that data is available in the mailbox. Secondly, the originator microcontroller receives an interrupt when data placed by that microcontroller in the mailbox has been received by the recipient microcontroller. This interrupt indicates that data has been received and that it is safe to write data to the mailbox.

The signals described above form the basis for clean and efficient data transfer between the two microcontrollers. The transfer time is limited only to the overhead of the interrupt service routines. The 5C060 can accept data at clock rates in excess of 20 MHz.

Programming the EPLDs

Figures 2 and 3 show the schematics and pin-out for the memory section, and Figure 4 is a schematic of the protocol sections in the mailbox memory. Using Intel's Programmable Logic Development System, these schematics can be transformed with ease into the logic equations that represent the desired function. The development system accepts a variety of entry methods, including schematic, netlist, state machine, and text file entry.

Once the design has been entered, the file is submitted to the Logic Optimizing Compiler (LOC), which performs an optional Boolean minimization, including De Morgan's inversion, and logically fits the design into the target EPLD.

The development system generates three output files. The Logic Equation File (LEF) contains the result of the minimization process, the Utilization Report File (RPT) contains the final device pinout, information about the internal logic routing, and a percent utilization for pins, macrocells, and product terms. Finally, the JEDEC file (JED) contains the device programming information required to program the EPLDs. These files are available from the authors.

Programming of the EPLDs is accomplished through Intel's Logic Programming Software (LPS) and the iUP-PC programming hardware. Designs also may be logically simulated through the use of Intel's FSIM software.

Summary

Applications such as industrial automation often require communication between multiple microcontrollers. Unfortunately this communication is hampered by the port orientation and lack of bus control signals within the microcontroller environment. One solution—as presented here—is the mailbox memory. The mailbox memory serves as an effective method for transferring data between microcontrollers, while the flexibility of the EPLDs serves as an effective way to implement the mailbox itself. □

ELECTRONIC DESIGN EXCLUSIVE

Regain lost I/O ports with erasable PLDs

Daniel E. Smith and Thomas B. Bowns
Intel Corp., 1900 Prairie City Rd., Folsom, CA 95630; (916) 351-2747.

As a means for reconstructing or regaining microcontroller I/O ports lost to memory expansion, erasable programmable logic devices, or EPLDs, contain all the necessary functions. In fact, EPLDs perform more functions than most programmable logic arrays, and offer the additional benefits of EPROM-like erasability, the low power consumption of CMOS technology, and gate densities near those of low-end gate arrays.

Lost I/O ports can be externally reconstructed with standard SSI packages. EPLDs, however, supply an alternative that reduces the external approach's impact on power and space consumption.

Erasable PLDs cut the space and power usually needed to reconstruct I/O ports. They can even build new ports, adding to a chip's capabilities.

The computing power of one-chip microcontrollers plays a role in many applications. But the growing complexity of these devices, as designers shift from 8- to 16-bit controllers, has strained their I/O capacity.

A typical 8-bit microcontroller in a 40-pin package contains a 4- to 8-kbyte program memory and 32 I/O pins, usually grouped into 8-bit ports. The 16-bit devices contain 8-kbyte memories and up to 40 I/O pins in packages that range from 48 to 68 pins. The possible number of ports falls short for some complex tasks in switching circuits, robotics, and automotive systems.

The I/O shortage is aggravated when the chip's internal program memory is too small for a given task. While tacking on external memory is easy enough, the addition consumes I/O pins.

Although some details vary, the basic techniques for reconstructing these lost I/O ports with EPLDs are the same for most microcontrollers. An example describes a 5C121 EPLD and an 8096 16-bit microcontroller, noting details specific to the microcontroller.

These techniques not only reconstruct ports on any available microcontroller, but they also are suited to adding new ports. For the 8096, the designer can add two new ports, 5 and 6, by changing to 1FFC-1FFF the hexadecimal address range in which the external memory is deselected. The new ports create a system with 56 I/O signals. The tradeoffs of this addition are the board space needed for two more EPLDs and two more bytes of reserved memory space at 1FFC and 1FFD.

The first consideration in reconstructing a port is the microcontroller's fixed-memory and I/O address map. In the 8096, memory-address ranges 0 to FF and 2000-3FFF contain on-chip registers, interrupt vectors, factory test code, and program memory. Expansion memory can go into the 100 to 1FFD range, a capacity of 8k bytes minus the first 256 and the last 2 bytes, and into the 4000 to FFFF range, another 8 kbytes.

The microcontroller has five 8-bit ports, three of which (0 to 2) are dedicated to I/O functions. Ports 3 and 4, however, are memory-mapped to 1FFE and 1FFF, respectively. These two ports reside right above the lower section of expansion memory space. (Other microcontrollers have the same functions, but their address ranges may vary.)

External memory, therefore, connects to the pins reserved for ports 3 and 4, eliminating them as general I/O ports. Reclamation of these ports calls for external latches and decode logic that disables the external memory and enables the latches at 1FFE and 1FFF. This logic decodes signals A_0 and Byte High Enable, $\overline{BHE}$, to select ports 3 and 4. The ports are selected either separately for 8-bit data transfers or together for 16-bit transfers.

The microcontroller multiplexes address and data on signal lines AD_0 to AD_{15}. As a result, Address Latch Enable, ALE, must latch the address as each bus cycle starts and keep it there for the cycle duration. Then the lines can transfer data throughout the cycle. Because $\overline{BHE}$ has the same timing as

"Reprinted with permission from Electronic Design (Vol. 35, No. 7) March 19, 1987. Copyright 1987 Hayden Publishing Co., Inc., a subsidiary of VNU."

the address, ALE must also latch $\overline{BHE}$.

Reconstruction of both ports without EPLDs requires 14 SSI packages if the high-current sink capability of open-collector drivers is needed. If not, nine packages will do.

Besides the address-decoding logic, the input ports need octal latches. The outputs contain octal latches, but inverting buffers are also needed. If the output does include open-collector drivers, the designer must add another set of inverting buffers to compensate for the drivers' inversion of the signal. In addition, a discrete flip-flop latches $\overline{BHE}$, and discrete gates decode the port selection and $\overline{RD}$ and $\overline{WR}$ signals.

On the other hand, reconstructing ports with EPLDs requires no logic outside of the EPLDs themselves (Fig. 1). Each device decodes its respective memory-mapped address, and one device disables the external memory at both 1FFE and 1FFF.

The EPLDs can sink 4 mA, which puts them in the same range as an SSI version without open-collector drivers. The designer can add open-collector drivers if a higher-current sink is needed.

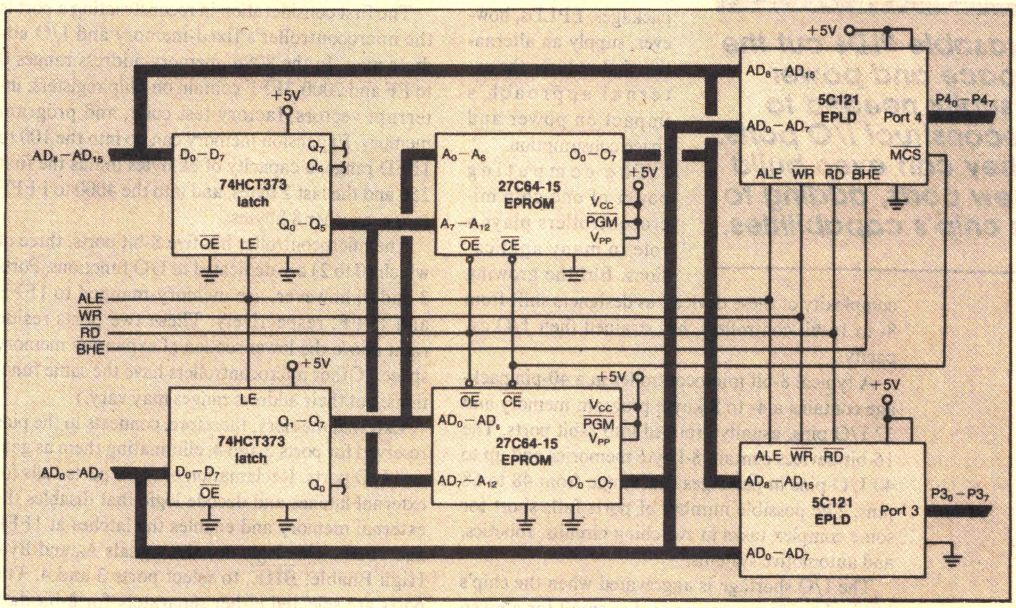
The design process leading to port reconstruction begins with defining the functions required of the EPLD and then creating a design file that can be translated into a Jedec file. Next, the designer programs the EPLD and tests the final circuit. Software can automate much of this procedure.

The first step is to list the functions the EPLD must perform. Then the designer identifies which EPLD feature best satisfies that need, because as with SSI logic, the device can accomplish its task in different ways.

In general, a device reconstructing a port must latch and decode address information from a multiplexed bus. The chip then produces an internal port-selection signal and an external memory-selection signal; the latter in address range 1FFE-1FFF. Moreover, the device acts as a bidirectional data path and decodes the $\overline{RD}$ and $\overline{WR}$ signals, routing the data with the port-selection signal (Fig. 2).

Drawing a schematic diagram of the EPLD helps isolate the circuit into functional blocks. In the example, combinatorial logic and three latches do the decoding at port 3.

Address lines AD_1 through AD_{11} pass through an AND gate and are latched as LAD_A. Address lines A_{12}



1. Two erasable programmable logic devices contain all the logic required to reconstruct ports 3 and 4 of an 8096 microcontroller. The two latches and two EPROMs comprise the external memory.

and inverted signals AD_{13} through AD_{15} pass through an AND gate and are latched as LAD_B. These two latched signals pass through another AND gate to create the Memory Disable Signal, MDS, which deactivates the EPROMs. Combined with LAD_0 (address signal AD_0 inverted and latched), LAD_A and LAD_B generate the port-selection signal.

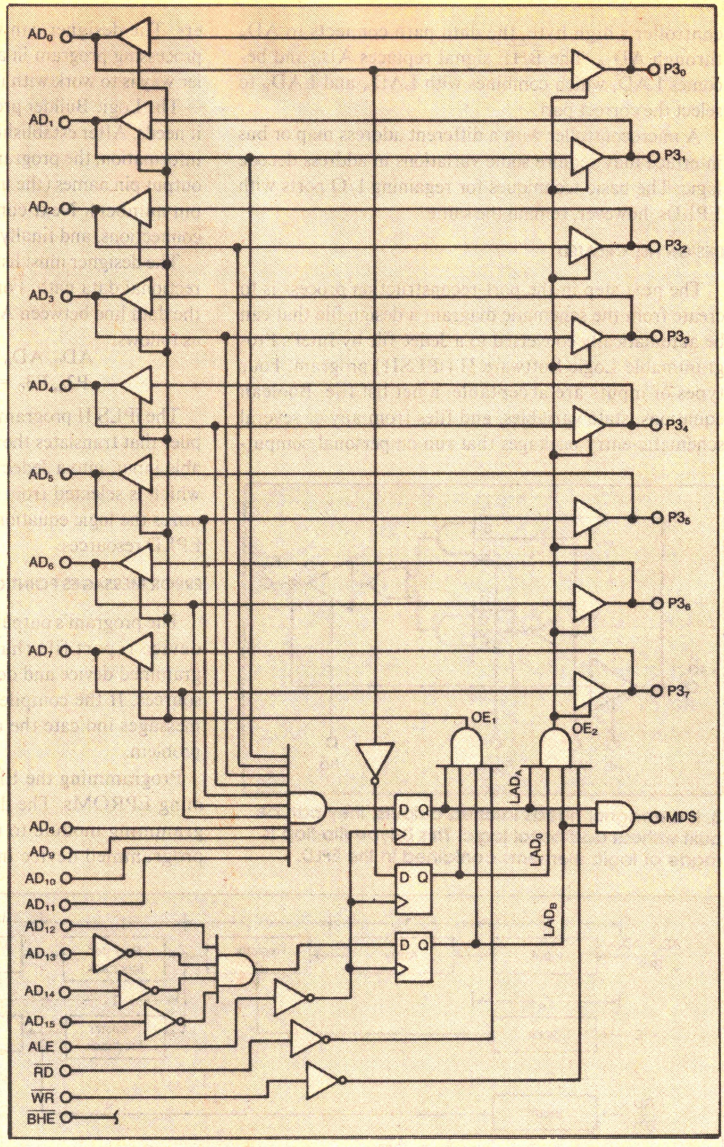
PARALLEL FORMAT SAVES TIME

The EPLD decodes and latches signals AD_1 through AD_{11} and AD_{12} through AD_{15} in parallel to minimize the time between address setup and ALE going low. An inverted ALE clocks the latches, which also store decoded addresses while the microcontroller transfers data over the bus.

Two combinatorial-output, internal feedback (COIF) primitives create a double-feedback loop with all output enables to the microcontroller bus controlled by OE_1, which is active during read operations. Output enables on the I/O side of the EPLD are controlled by OE_2, which is active during write operations. Thus data is valid at the inputs or outputs only while the appropriate command, $\overline{RD}$ or $\overline{WR}$, is active.

If the application calls for latched outputs, the designer can create them from logic on the EPLD. One configuration is a D-type latch activated by the trailing edge of WR (Fig. 3). In this circuit, the outputs are always enabled, except during reads, when they are placed in a high-impedance state. The Reset signal clears the outputs to a logic 0 during initialization.

The fourth port's schematic varies little from that of the third. Because port 4 handles data transfers on the micro-



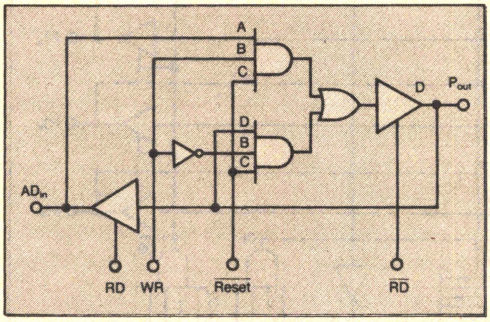
2. The schematic for the port 3 EPLD contains a bidirectional path that includes parallel address decoding that speeds circuit operation. In the port 3 device, A_0 is inverted and latched, then used to qualify reads and writes; the port 4 EPLD relies on BHE for qualification.

controller's high byte, the data path connects to AD_8 through AD_{15}. The $\overline{BHE}$ signal replaces AD_0 and becomes LAD, which combines with LAD_A and LAD_B to select the correct port.

A microcontroller with a different address map or bus interface may require some variations in address decode logic. The basic techniques for regaining I/O ports with EPLDs, however, remain the same.

DESIGN FILE CREATED

The next step in the port-reconstruction process is to create from the schematic diagram a design file that can be automatically converted to a Jedec file by Intel's Programmable Logic Software II (iPLSII) program. Four types of inputs are acceptable: a net list file, Boolean equations, state variables, and files from any of several schematic-entry packages that run on personal computers. The designer can write a net list file with a word-processing program in a nondocument mode, but an easier way is to work with iPLS II's Logic Builder.



3. If a designer needs latched outputs, they can be built without additional logic. This D-type flip-flop is made of logic elements contained in the EPLD.

The Logic Builder prompts the user for the information it needs. After establishing the file with some background information, the program asks for lists of all the input and output pin names (the user can assign a name to a specific pin number). Next come the internal assignments and connections, and finally, the logic equations needed.

The designer must list all the COIFs that form the bidirectional data path. For example, the entries that create the data line between AD_0 and $P3_0$ (see Fig. 2 again) are as follows:

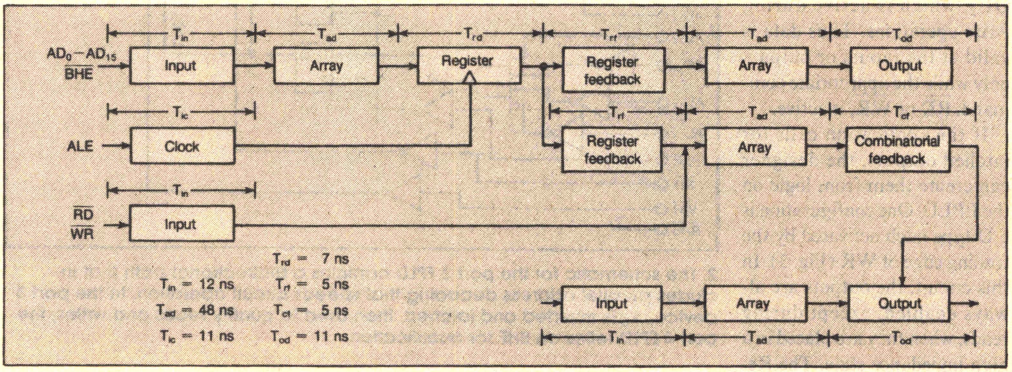
$$AD_0, AD_0 = COIF\,(P3_0, OE_1)$$
$$P3_0, P3_0 = COIF\,(AD_0, OE_2)$$

The iPLS II program contains a logic-optimizing compiler that translates the schematic's net list, or other suitable input, into a Jedec programming file. The compiler, which is selected from the program's main menu, optimizes the logic equations and assigns I/O pins and other EPLD resources.

ERROR MESSAGES POINT OUT PROBLEM

The program's outputs are the programming file and a device report file that shows the pinout of the programmed device and describes the use of the device's resources. If the compiler cannot translate the file, error messages indicate the design-file entry that caused the problem.

Programming the EPLD is very similar to programming EPROMs. The designer connects an EPLD programming module to the workstation, inserts an unprogrammed device into the socket, and calls up the



4. A block diagram of an ELPD's internal delays shows how users can determine the maximum delay for each signal path and, as a result, the port's maximum operating frequency.

programming menu. The menu asks for the device's type and the Jedec file name, and the system then programs and verifies the chip.

Considering how straightforward the port-reconstruction functions are, the best test of the programmed EPLD is to plug it into a circuit and see if it works. An EPROM-based microcontroller with some simple read and write routines to exercize the device works well. The designer can also use an in-circuit emulator for the microcontroller, if one is available.

Any bugs can be fixed quickly. To correct a bug the user erases the EPLD file and changes the design file, which then can be recompiled and the device reprogrammed.

A timing analysis confirms the EPLD's compatibility with different microcontroller clock speeds. The analysis amounts to adding the internal delays for paths through the EPLD and comparing these path delays to the microcontroller's timing requirements.

The three paths of interest are Address Setup to ALE, which must take no longer than 116 ns for an 8096 operating at 6 MHz; and no longer than 50 ns at 10 MHz. Other maximum values are: Data Valid From $\overline{RD}$, 358 ns and 230 ns; and Data Valid Before Write, 272 ns and 130 ns.

A block diagram of the specific device with each internal delay is needed for the timing analysis. For the example circuit, the Address Setup to ALE delay for the port 3 EPLD is 49 ns (Fig. 4). This value, achieved by decoding and latching AD_1 to AD_{11} in parallel with AD_{12} to AD_{15}, just meets the maximum delay at 10 MHz.

The delay for Data Valid From $\overline{RD}$ is the sum of delays in the enable path and the data path, or 136 ns. The delay path for the write operations is shorter: It is that for the enable path added to 41 ns for the data path (after eliminating a 30-ns overlap in enable and data timing), or 106 ns. Both are well within limits. □

Daniel E. Smith, a senior technical writer at Intel, has also worked in microcomputer-systems testing and written manuals for microprocessors, development software, and bubble memories. He has a BA in history from San Jose University and an MA in biblical studies from the Graduate Theological Union/Jesuit School of Theology in Berkeley, Calif.

Thomas B. Bowns is an application engineer for Intel's EPLD operation. He also has worked as a technician on the company's EPROM line. Bowns studied digital and microwave electronics at American River College in Carmichael, Calif.

Advanced Architecture EPLDs

3

3 Advanced Architecture EPLDs

PRELIMINARY

5CBIC PROGRAMMABLE BUS INTERFACE CONTROLLER

- **Higher Integration Alternative to Transceivers, Latches, Multiplexers and PAL* Functions**
- **Applications Include Dual Port Control, Multiplexed Bus Interface, DRAM Control and Similar Functions**
- **Port-Oriented Bus Management Unit Supports:**
  - **— 3-Way Asynchronous Data Transfer on Byte-Wide Buses**
  - **— Programmable Option of Latched or Real Time Data**
  - **— True or Complement Data Path**
- **Macrocell-Based Programmable Logic Unit Provides:**
  - **— Variable Input and Output Architecture**
  - **— On-Chip Controls for the Bus Management Unit**
  - **— Up to Eight Buried Registers**
  - **— Programmable Registers can be Configured as Positive Edge-Triggered D-, J-K, R-S or T- Types**
  - **— Asynchronous Preset and Clear on All Registers**
  - **— Option of Latched Inputs**
- **Low Power: 75 μA Typical Standby**
- **CHMOS EPROM Technology Based:**
  - **— Max Bus Port Drive Capability: 16 mA**
  - **— Typical Data Transfer Delay Between Ports = 25 ns**
  - **— Logic Array Operating Frequency = 20 MHz**
- **Available in 44-Lead PLCC Package**
  (See Packaging Spec., Order # 231369)

The Intel 5CBIC is useful in implementing bus interfacing logic functions that have traditionally been done using SSI/MSI TTL components. Core bus functions are provided that can be customized using EPROM bits for specific applications. Control logic can also be implemented through a sum of products architecture that is included in this 44-lead PLCC package. Such levels of integration are realized utilizing the benefits of Intel's advanced CHMOSII-E process.

This general purpose architecture is supported by iPLDS II, Intel's Programmable Logic Development System, to develop the design and program the devices. Several methods of entry facilitate the design resulting in shorter completion times.

*PAL is a trademark of Monolithic Memories, Inc.

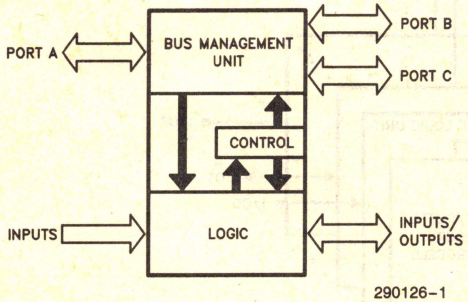


Figure 1. Block Diagram

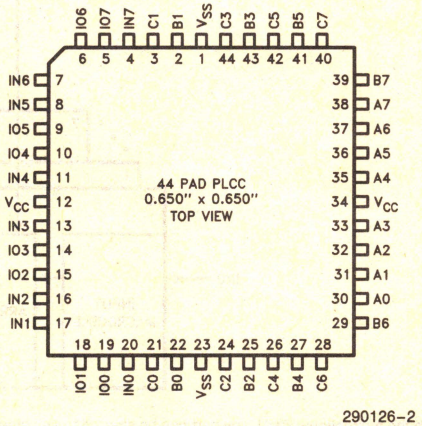


Figure 2. Lead Configuration

November 1987
Order Number: 290126-003

FUNCTIONAL DESCRIPTION

As the name suggests, this programmable bus interface controller offers a high integration solution to design problems involving data transfer on bus lines and the logic needed to control these transfers. This integration directly translates into savings in board space and lower system cost for equivalent functions implemented using conventional SSI/MSI components.

Present in the port-oriented 5CBIC are two functional blocks that enable complex bus functions to be realized: the Bus Management Unit (BMU) and the Programmable Logic Unit (PLU). These two units communicate with each other through the input and the feedback buses. A control section shown in Figure 3 steers signals from the PLU to the two units through the control bus.

ARCHITECTURE DESCRIPTION

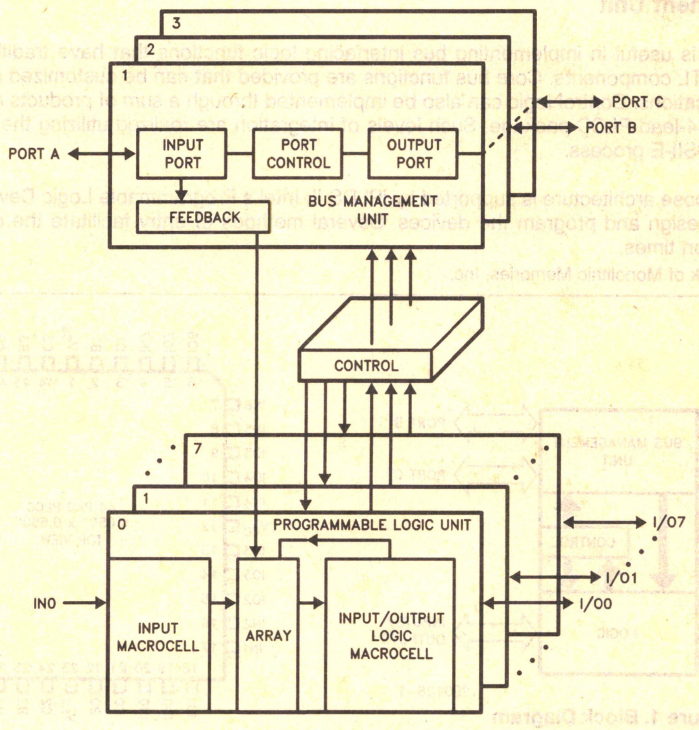
The innovative architecture of the 5CBIC incorporating a port-oriented approach for bus interface control is illustrated in Figure 5. The Bus Management Unit (BMU) and the Programmable Logic Unit (PLU) interface to the feedback and the control busses. The macrocells in the PLU feed the input bus.

Bus Management Unit (BMU)

The Bus Management Unit (BMU) comprises three ports: PA, PB and PC (Figure 4a). Each of these ports is bidirectional and 8 bits wide. Data can be routed from any port to any other port.

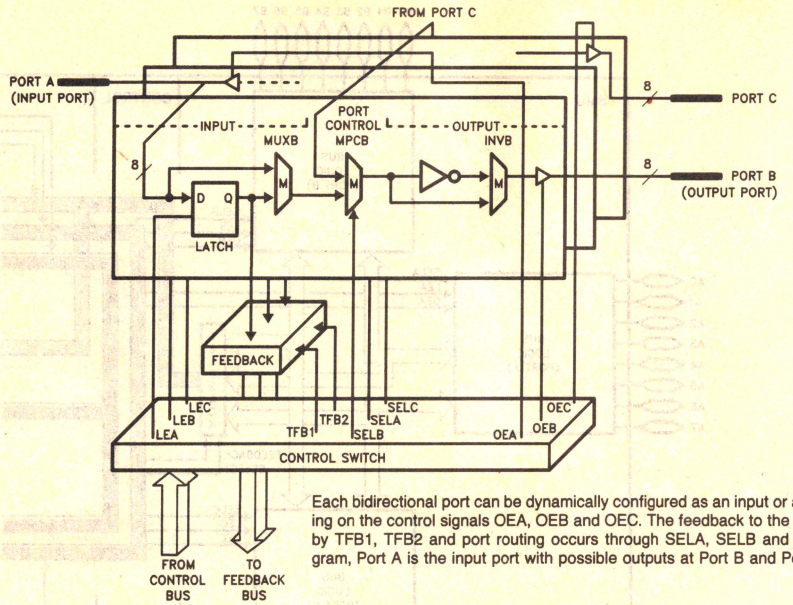
Data into any port can be user-selected to be latched by a port Latch Enable signal, (LE). Routing of latched or unlatched data between ports is achieved using a combination of EPROM architecture and dynamic control signals defined by the user. Data out of any port can be programmed to have an inverted sense through EPROM architecture control (INV).

Each bidirectional port can be dynamically configured as an input or an output depending on the control signals OEA, OEB and OEC. Latched data from



In the tridirectional BMU, any port can be steered to any other port. In this diagram, Port A can be directed to Port B or Port C or both. The PLU provides a 600-gate equivalent PAL function.

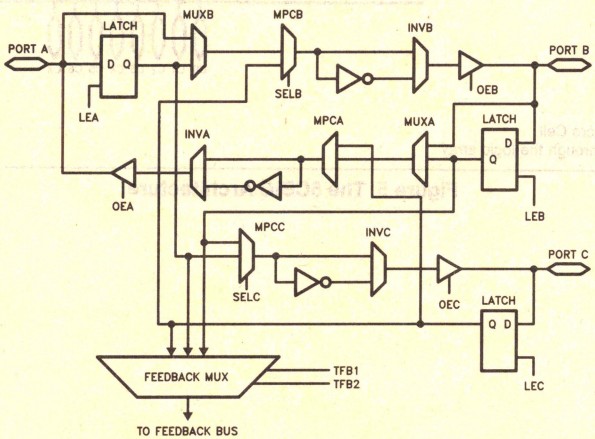
Figure 3. Functional Blocks in the 5CBIC



Each bidirectional port can be dynamically configured as an input or an output depending on the control signals OEA, OEB and OEC. The feedback to the array is controlled by TFB1, TFB2 and port routing occurs through SELA, SELB and SELC. In the diagram, Port A is the input port with possible outputs at Port B and Port C.

290126-5

Figure 4a. Bus Management Unit Block Diagram



290126-23

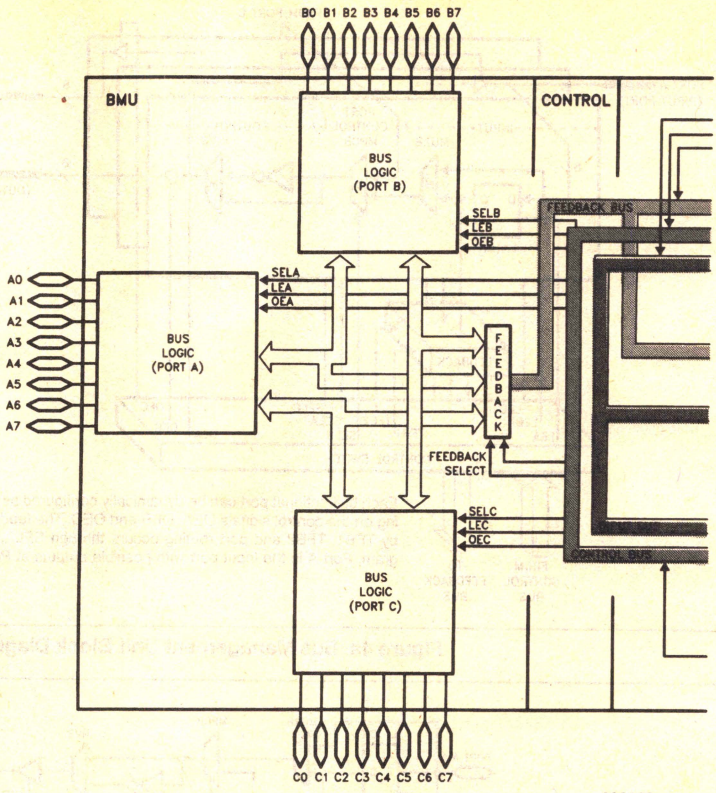
LEGEND:
OEA, OEB, OEC, SELA, SELB, SELC, LEA, LEB, LEC, TFB1 and TFB2 are the control outputs for the BMU derived from the control bus.
MPCA, MPCB and MPCC are dynamic multiplexers controlled by SELA, SELB and SELC for port selection.
MUXA, MUXB, INVA, INVB and INVC are static multiplexers controlled by architecture bits (EPROM bits).
All latches are the "transparent" type.

Figure 4b. BMU Logic Diagram

any incoming port can be fed internally to the array through TFB1 and TFB2. The three ports can be time-multiplexed, if needed. Port routing is controlled by signals SELA, SELB and SELC (Figure 4b).

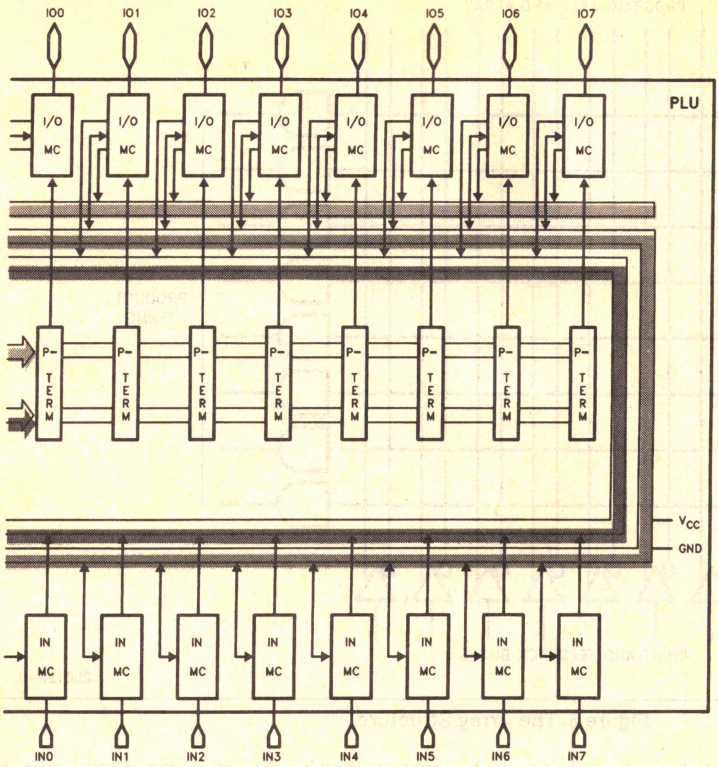
Programmable Logic Unit (PLU)

An on-chip 600-gate-equivalent EPLD supplies the control signals to the bus unit and related applica-



INMC = Input Macro Cell
IOMC = Input/Output Macro Cell
p-term = Product Terms through the logic array

Figure 5. The 5CBIC Architecture


IO0
IO1
IO2
IO3
IO4
IO5
IO6
IO7
PLU
I/O MC
P-TERM
VCC
GND
IN MC
IN0
IN1
IN2
IN3
IN4
IN5
IN6
IN7

290126-21

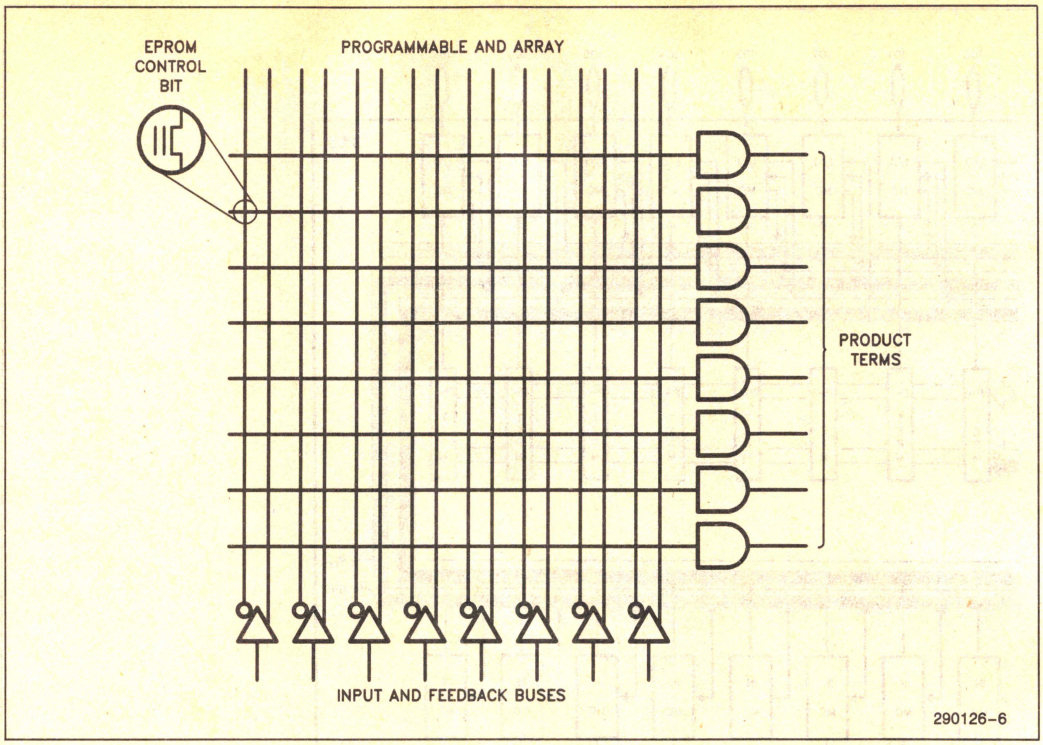


Figure 6. The Array Structure

tion functions in the system. A dedicated input port and a bidirectional I/O port, each 8 bits wide, allows control logic implementation in the 5CBIC. The macrocell based architecture enables the designer to use up to 24 inputs and 8 outputs.

The inputs, array and I/O marcrocells generate a sum-of-products (AND-OR) representation of any given logic. Within the AND array, there is an EPROM connection at every intersection of an incoming signal (true and complement) and a product term to a given macrocell (Figure 6). Before programming an erased device an EPROM connection exists at every intersection. It is during the programming process that these connections are opened to generate the required connections.

The bidirectional I/O port, when configured as an input, is identical to the input port in that inputs may be latched by a signal from the control bus as shown in Figure 7. An additional flow-through option for the data inputs is available in the input macrocell.

The variable output architecture in the PLU allows the designer to select the combinatorial or registered output types on a macrocell basis. This may be implemented by selecting the architecture bit MARB1 and the edge-triggered flip-flop (Figure 7). The Macrocells support D, T, S-R or J-K type registers for optimal design. Truth tables for these are listed in Figure 8 for easy reference. Whereas all eight of the product terms are OR-ed together at the register input for the D- and the T- registers, the J-K and the S-R configurations employ sharing of the product terms among two OR-gates.

The registers receive inputs at its data, clock, set and reset lines. Eight product terms are available for the data input and one each for the set and the clear inputs.

The clock, output enable and the latching signals can be selected by architecture bits MARB2, 6 and 3 respectively to be outputs from the control bus or one product term from the array. Designers thus have more options available for asynchronous clocking and output controls.

The macrocell output can be fed back to the array through the feedback bus or to the control bus. Figure 9 summarizes the bus structure and its relationship to the relevant units in the 5CBIC.

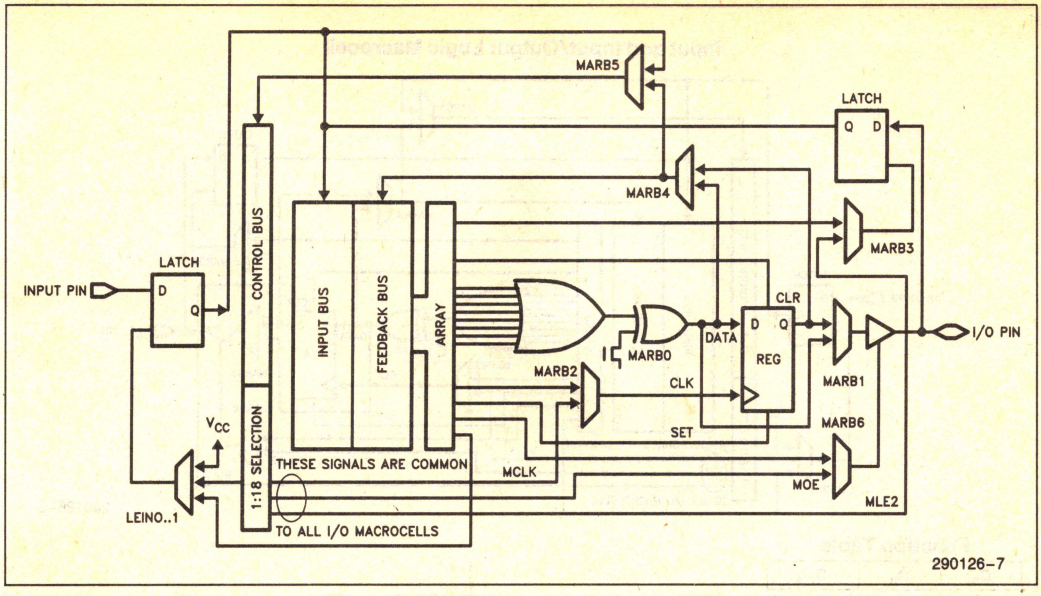


Figure 7. The Programmable Logic Unit

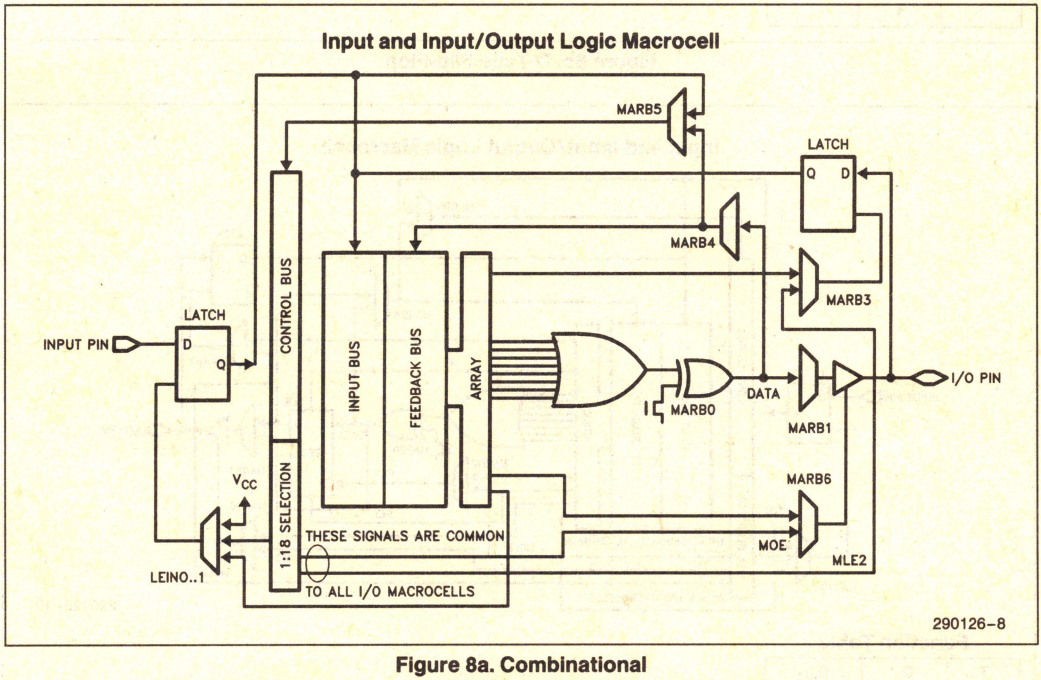
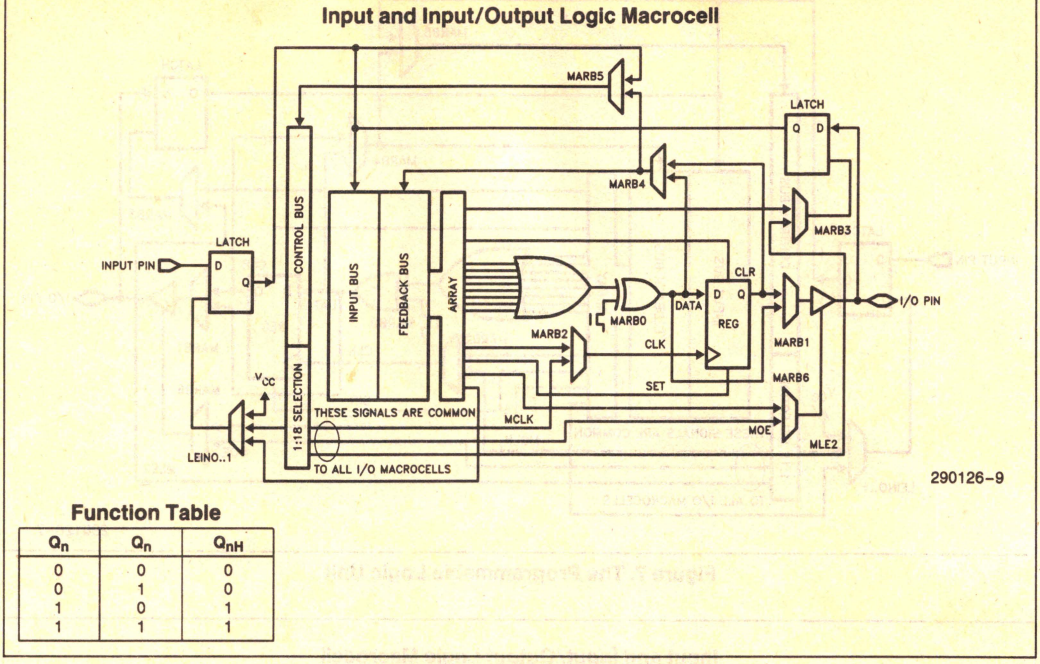


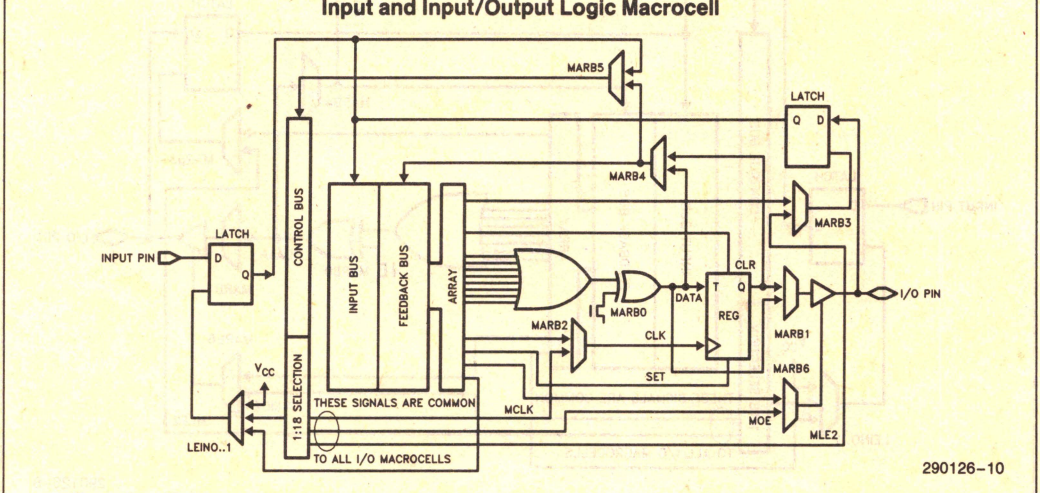
Figure 8a. Combinational



Function Table

Q_n	Q_n	Q_{nH}
0	0	0
0	1	0
1	0	1
1	1	1

Figure 8b. D-Type Flip-Flop

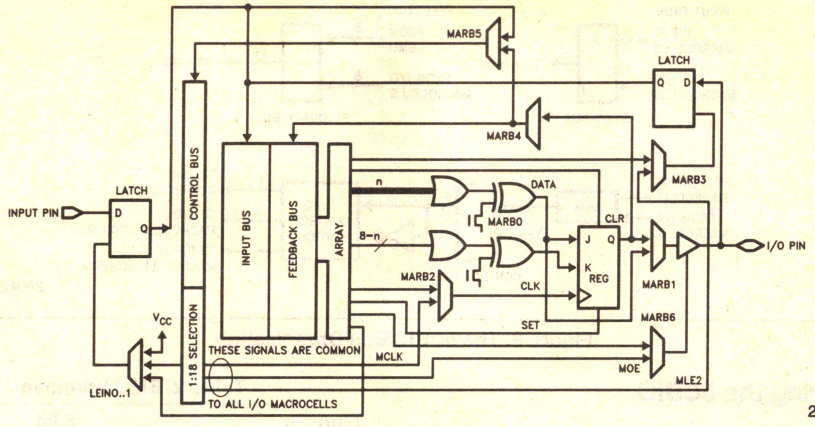


Function Table

T	Q_n	Q_{nt1}
0	0	0
0	1	1
1	0	1
1	1	0

Figure 8c. Toggle Flip-Flop

Input and Input/Output Logic Macrocell



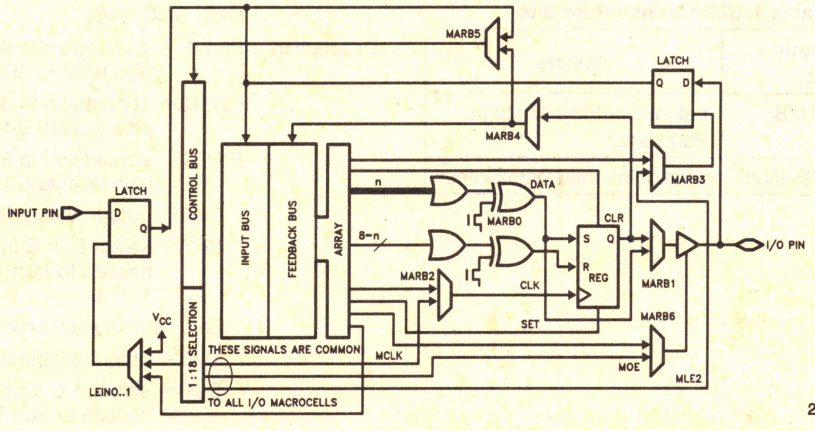
290126–11

Function Table

J	K	Q_n	$Q_n + 1$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Figure 8d. J-K Flip-Flop

Input and Input/Output Logic Macrocell



290126–12

Function Table

S	R	Q_n	$Q_n + 1$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	Illegal	

Figure 8e. S-R Flip-Flop

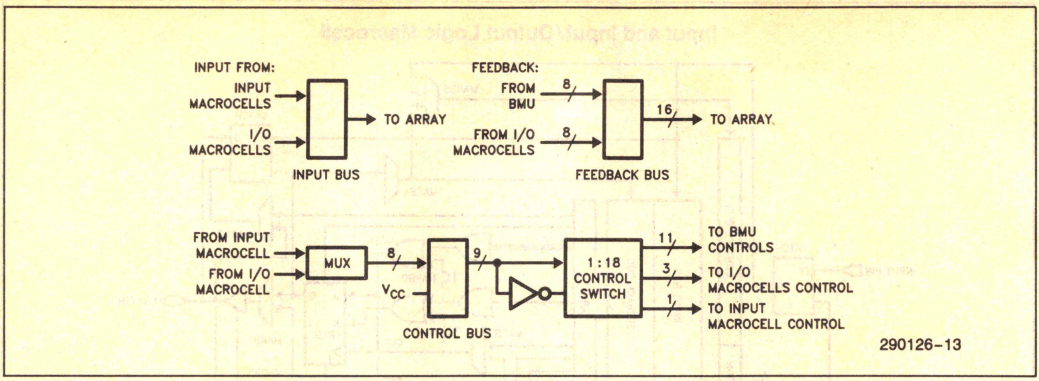


Figure 9. The 5CBIC Bus Organization

Configuring the 5CBIC

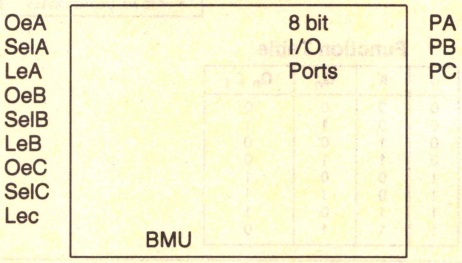
The Device Configuration Manager (DCM) in iPLS II provides a high-level graphic design entry alternative that allows bus configurations to be implemented in minutes. A more detailed explanation is given in the iPLS II manual. An ADF (Advanced Design File) is then automatically generated that defines the logic network using primitives.

The primitive necessary for configuring inter-port communication is the "BMU", while the one required for internal feedback from the BMU to the PLU is the feedback primitive "BFMUX". Tables 1 through 4 define these primitives and their fields/bits.

Table 1. BMU Architecture Bits

Architecture Bit	Selects
MUXA, MUXB	Latched or Flow-Through Port Data
INVA, INVB, INVC	True or Inverted Data Output

Table 2. BMU Primitive



Name: BMU (Bus Management (Unit)

ADF Syntax: PortA, PortB, PortC = BMU (Type, OeA, SelA, LeA, OeB, SelB, LeB, OeC, SelC, LeC)

Description:
- Port A = connection to 8 parallel I/O pins labeled A0–A7
- Port B = connection to 8 parallel I/O pins labeled B0–B7
- Port C = connection to 8 parallel I/O pins labeled C0–C7
- OeA = output enable for Port A
- SelA = select B or C internal connection to Port A (0 = C, 1 = B)
- LeA = input latch enable for Port A
- OeB = output enable for Port B
- SelB = select A or C internal connection to Port B (0 = C, 1 = A)
- LeB = input latch enable for Port B
- OeC = output enable for Port C
- SelC = select A or B internal connection to Port C (0 = A, 1 = B)
- LeC = input latch enable for Port C

Inversion Control				Input Latch		
Port:	**A**	**B**	**C**	**A**	**B**	**C**
Bit:	**5**	**4**	**3**	**2**	**1**	**0**
0	Invert Output	Invert Output	Invert Output	Latched A	Latched B	Latched C
1	No Invert	No Invert	No Invert	Direct A	Direct B	Latched C*

*If LeC is continually high, the C latch is transparent.

Table 3. Bus Feedback Multipler Primitive

BFMX

TFB1	0	0	1	Fbk
TFB2	0	1	0	[0:7]
	C	B	A	

Name: BFMX (Bus Feedback Multiplexer

ADF Syntax: Fbk[0:7] = BFMX (TFB1, TFB2)

Description: Outputs.

Fbk = 8 parallel lines of feedback to logic array.

Inputs:

TFB1, TFB2 = By appling 0 or 1 as shown on the chart above, select feedback from Port A, B, or C. TFB1 and TFB2 can be set to VCC or GND, or they can be connected to any internal feedback or input node. The ports are defined in the BMU primitive section.

Table 4. PLU Architecture Bits

Architecture Bit	Selects
MARB0	Output Polarity
MARB1	Combinatorial or Registered Outputs
MARB2	Clock Source
MARB3	Latching Signal Source
MARB4	Combinatorial or Registered Feedback to the Logic Array
MARB5	Input Source to the Control Bus
MARB6	tri-state Control Signal

ABSOLUTE MAXIMUM RATINGS*

Symbol	Parameter	Min	Max	Units
V_{CC}	Supply Voltage(1)	−2.0	7.0	V
V_{PP}	Programming Supply Voltage(1)	−2.0	13.5	V
V_I	DC Input Voltage(1)(2)	−0.5	V_{CC} + 0.5	V
t_{stg}	Storage Temperature	−65	+150	°C
t_{amb}	Ambient Temperature(3)	−10	+85	°C

NOTES:
1. Voltages with respect to ground.
2. Minimum DC input is −0.5V. During transitions, the inputs may undershoot to −2.0V for periods less than 20 ns under no load conditions.
3. Under bias. Extended temperature versions are also available.

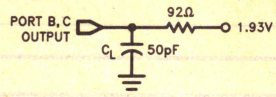
**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

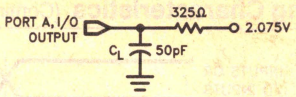
D.C. CHARACTERISTICS T_A = 0°C to +70°C, V_{CC} = 5.0V ±5%

Parameter	Description	Min	Max	Unit	Test Conditions
V_{OH}	Output High Voltage	2.4		V	TTL: I_{OH} Port A −1 mA, Port B, C −5 mA, I, I/O −1 mA, V_{CC} = Min
V_{OL}	Output Low Voltage		0.45	V	I_{OL} Port A 5 mA, Port B, C 16 mA, I, I/O 5 mA, V_{CC} = Min
V_{IH}	Input High Level	2.0	V_{CC} + 0.3	V	
V_{IL}	Input Low Level	−0.3	0.8	V	
I_I	Input Leakage Current		10	µA	$V_{SS} \le V_{IN} < V_{CC}$, V_{CC} = Max
I_{OZ}	Output Leakage Current		10	µA	$V_{SS} \le V_{OUT} \le V_{CC}$, V_{CC} = Max
I_{OS}(4)	Output Short Circuit Current	BMU PLU	80 16	mA mA	V_{CC} = Max, V_{OUT} = 0.5
I_{SB}(5)	Operating Current (standby, low power mode)		75	µA	V_{IN} = V_{CC} or Gnd, I_O = 0
I_{CC2}	Operating Current (active, low power mode)		20	mA	V_{IN} = V_{CC} or Gnd, f = 1 MHz, No Load
I_{CC3}	Operating Current (active, turbo mode)		108	mA	V_{IN} = V_{CC} or Gnd, f = 1 MHz, No Load
C_{IN}	Input Pin Capacitance		30	pF	
C_{OUT}	Output Pin Capacitance		40	pF	

NOTES:
4. Output shorted for no more than 1 sec. and only one output shorted at a time.
5. Chip automatically goes into standby mode if logic transitions do not occur at input pins. (Approximately 100 ns after last transition).



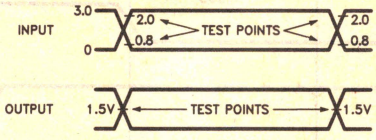
290126-14



290126-15

NOTES:
1. C_L includes jig capacitance
2. Device input rise and fall times < 6 ns

Figure 10A. A.C. Testing Load Circuit



290126-16

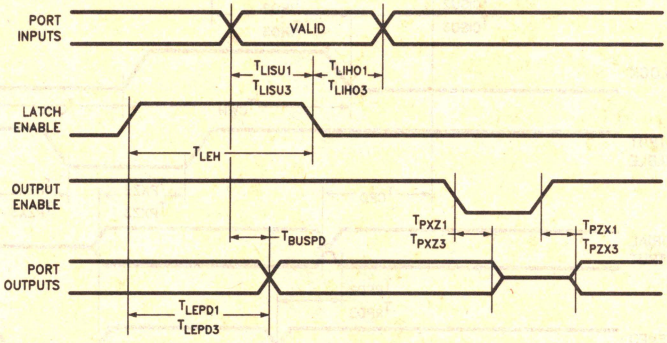
A.C. Testing: Inputs are driven at 3.0V for a Logic "1" and 0V for a Logic "0". Timing measurements are made at 2.0V for a Logic "1" and 0.8V for a Logic "0" on inputs. Outputs are measured at a 1.5V point.

Figure 10B. A.C. Testing Input, Output Waveform

Switching Characteristics

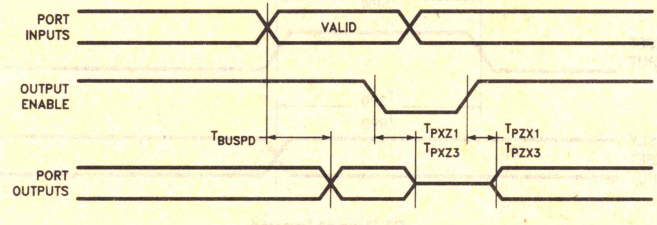
Notation:

Timing Suffix	Referenced to Control From:
1	direct input pin
2	product term
3	control bus



290126-17

A) Latched Port Inputs

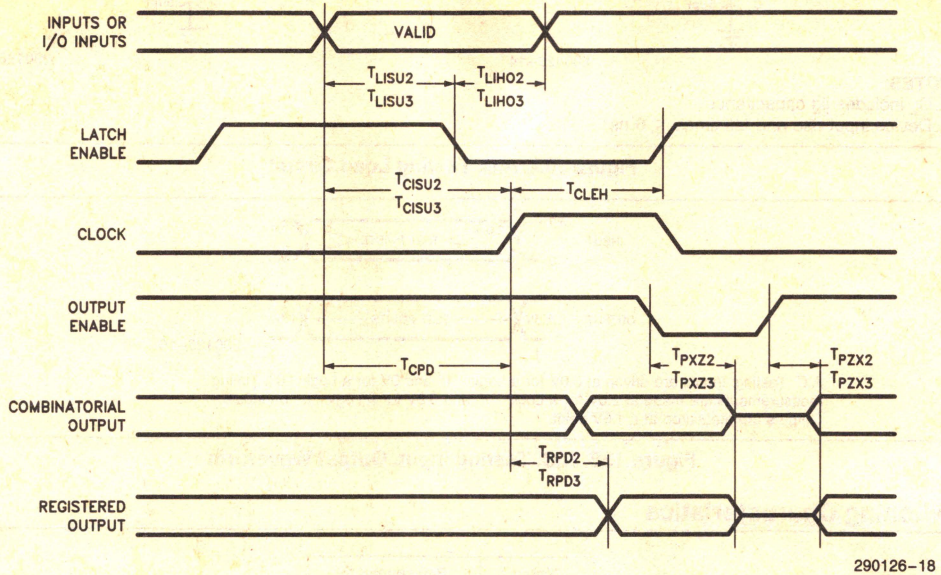


290126-22

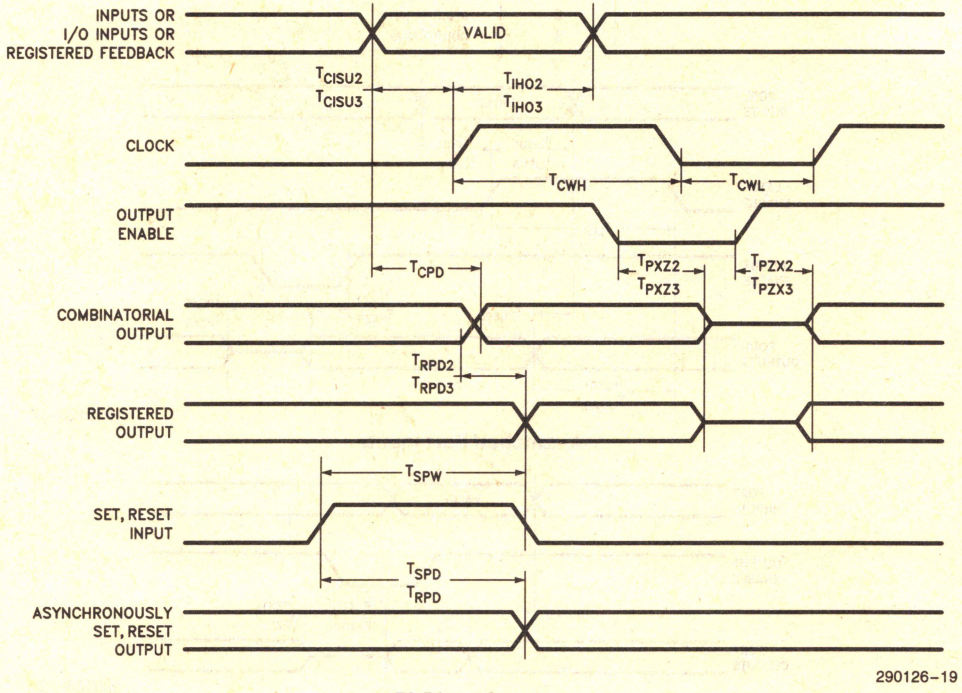
B) Direct Port Inputs

Figure 10C. Bus Management Unit

Switching Characteristics (Continued)



A) Latched Inputs



B) Direct Inputs

Figure 10D. Programmable Logic Unit

AC CHARACTERISTICS

BUS MANAGEMENT UNIT

Symbol	Parameter	−25			−35			−45			Units
		Min	Typ	Max	Min	Typ	Max	Min	Typ	Max	
T_{LISU1}	Port Input Setup Time to Latch Enable (Fast Option)	0			0			0			ns
T_{LISU3}	Port Input Setup Time to Latch Enable (Control Bus)	0			0			0			ns
T_{LIHO1}	Port Input Hold Time to Latch Enable (Fast Option)	25			35			45			ns
T_{LIHO3}	Port Input Hold Time to Latch Enable (Control Bus)	75			85			95			ns
T_{LEH}	Latch Enable High Time	25			35			45			ns
T_{BUSPD}	Port to Port Propagation Delay		15	25		25	35		35	45	ns
T_{PXZ1}	Valid Output to High Impedance (OE From Fast Option)		15	25		25	35		35	45	ns
T_{PXZ3}	Valid Output to High Impedance (OE From Control Bus)			75			85			95	ns
T_{PZX1}	High Impedance to Valid Output (OE From Fast Option)		15	25		25	35		35	45	ns
T_{PZX3}	High Impedance to Valid Output (OE From Control Bus)			75			85			95	ns
T_{LEPD1}	Latch Enable (From Fast Option) To Port Output Delay		15	25		25	35		35	40	ns
T_{LEPD3}	Latch Enable (From Control Bus) To Port Output Delay			75			85			95	ns

PROGRAMMABLE LOGIC UNITS

Symbol	Parameter	−25			−35			−45			Units
		Min	Typ	Max	Min	Typ	Max	Min	Typ	Max	
T_{LISU2}	Input Setup Time to Latch Enable (P-Term)	0			0			0			ns
T_{LISU3}	Input Setup Time to Latch Enable (Control Bus)	0			0			0			ns
T_{LIHO2}	Input Hold Time to Latch Enable (P-Term)	50	30		70	50		80	60		ns
T_{LIHO3}	Input Hold Time to Latch Enable (Control Bus)	70	50		80	60		90	70		ns
T_{CISU2}	Input Setup Time to Clock (P-Term)	0			0			0			ns
T_{CISU3}	Input Setup Time to Clock (Control Bus)	0			0			0			ns
T_{CLEH}	Clock to Latch Enable Hold Time	5			5			5			ns
T_{CPD}	Combinatorial Output Delay		40	65		50	75		60	85	ns

PROGRAMMABLE LOGIC UNITS (Continued)

Symbol	Parameter	−25			−35			−45			Units
		Min	Typ	Max	Min	Typ	Max	Min	Typ	Max	
T_{RPD2}	Registered Output from Clock (P-Term)		20	30		30	40		40	45	ns
T_{RPD3}	Registered Output from Clock (Control Bus)		30	35		30	40		40	45	ns
T_{IHO2}	Input Hold Time to Clock (P-Term)	50	30		70	50		80	60		ns
T_{IHO3}	Input Hold Time to Clock (Control Bus)	70	50		80	60		90	70		ns
T_{CWH}	Minimum Clock Width High	33			38			43			ns
T_{CWL}	Minimum Clock Width Low	33			38			43			ns
T_{SPD}	Set Output Delay			65			75			85	ns
T_{RPD}	Reset Output Delay			65			75			85	ns
T_{SPW}	SET/RESET Pulse Width	33			38			43			ns
T_{PXZ2}	Valid Output to High-Impedance (OE from P-Term)			65			75			85	ns
T_{PXZ3}	Valid Output to High Impedance (OE from Control Bus)			75			85			95	ns
T_{PZX2}	High Impedance to Valid Output (OE from P-Term)			65			75			85	ns
T_{PZX3}	High Impedance to Valid Output (OE from Control Bus)			75			85			95	ns
T_{CP1}	Minimum Clock Period (Register Output to Register Input Through Feedback Path)			50			70			80	ns
F1	Maximum Internal Frequency	20.0			14.3			12.5			MHz
T_{CP2}	Minimum Clock Period Between Logic Transitions (Inputs to Outputs)		50	80		80	110		100	120	ns
F2	Maximum External Frequency	12.5	20.0		9.09	12.5		8.0	10.0		MHz

int$_e$ligent Programming Algorithm™

The 5CBIC supports the int$_e$ligent Programming Algorithm which rapidly programs Intel H-ELPDs (and EPROMs) using an efficient and reliable method. The int$_e$ligent Programming Algorithm is particularly suited to the production programming environment. This method greatly decreases the overall programming time while programming reliability is ensured as the incremental program margin of each bit is continually monitored to determine when the bit has been successfully programmed.

FUNCTIONAL TESTING

Since the logical operation of the 5CBIC is controlled by EPROM elements, the device is completely testable. Each programmable EPROM bit controlling the internal logic is tested using application-independent test program patterns. After testing, the devices are erased before shipment to customers. No post-programming tests of the EPROM array are required.

The testability and reliability of EPROM-based programmable logic devices is an important feature over similar devices based on fuse technology. Fuse-based programmable logic devices require a user to perform post-programming tests to insure proper programming.

DESIGN SECURITY

A single EPROM bit provides a programmable design security feature that controls the access to the data programmed into the device. If this bit is set, a proprietary design within the device cannot be copied. This EPROM security bit enables a higher degree of design security than fused-based devices since programmed data within EPROM cells is invisi-

ble even to microscopic evaluation. The EPROM security bit, along with all the other EPROM control bits, will be reset by erasing the device.

TURBO-BIT

The device will consume quiescent current (75 μA, typically) if no transitions are detected in the array for 50 ns or more. This mode, the power-down mode, can be enabled by selecting the Turbo Bit OFF. If this bit is enabled, however, the device consumes active current. The power-down mode will revert to its active state if a transition is detected in the array, at an extra delay of 3 ns in speed paths.

LATCH-UP IMMUNITY

All pins of the 5CBIC have been designed to resist latch-up which is inherent in inferior CMOS structures. The 5CBIC designed with Intel's proprietary CHMOS II-E EPROM process. Thus, pins will not experience latch-up with currents up to 100 mA and voltages ranging from $-1V$ to $V_{CC} + 1V$. Furthermore, the programming pin is designed to resist latch-up to the 13.5V maximum device limit.

INTEL PROGRAMMABLE LOGIC DEVELOPMENT SYSTEM (iPLDS II)

The iPLDS II graphically shown in Figure 11 provides all the tools needed to design with Intel H-Series EPLDs or compatible devices. In addition to providing development assistance, iPLDS II insulates the user from having to know all the intricate details of EPLD architecture (the machine will optimize a design to benefit from architectual features). It contains comprehensive third generation software that supports four different design entry methods, minimizes logic, does automatic pin assignments and produces the best design fit for the selected EPLD. It is user friendly with guided menus, on-line Help messages and soft key inputs.

In addition, the iPLDS II contains programmer hardware in the form of an iUP-PC Universal Programmer-Personal Computer to enable the user to program EPLDs, read and verify programmed devices and also to graphically edit programming files. The software generates industry standard JEDEC object code output files which can be downloaded to other programmers as well.

The iPLDS II has interfaces to popular schematic capture packages (including Dash series from Futurenet* and PC CAPS from PCAD**) to enable designs to be entered using schematics. A more integrated schematic entry method is provided by SCHEMA II-PLD, a low-cost schematic capture package that supports EPLD primitives and user-defined macro symbols. SCHEMA II-PLD contains the EPLD Design Manager, which provides a single user interface to both SCHEMA II-PLD and iPLS II software. The other design formats supported are Boolean equation entry and State Machine design entry.

The iPLDS II operates on the IBM† PC.XT, PC/AT, or other compatible machine with the following configuration:

1. At least one floppy disk drive and hard disk drive.
2. MS-DOS†† Operation System Version 3.0 or greater.
3. 512K Memory.
4. Intel iUP-PC Universal Programmer-Personal Computer
5. A GUPI LOGIC Adaptor
6. A color monitor is suggested.

Detailed information on the Intel Programmable Logic Developement System is contained in a separate Intel data sheet.

*FutureNet is a registered trademark of FutureNet Corporation. DASH is a trademark of FutureNet Corporation.

**PC-CAPS is a trademark of P-CAD Corporation.

†IBM Personal Computer is a registered trademark of International Business Machines Corporation.

††MS-DOS is a registered trademark of Microsoft Corporation.

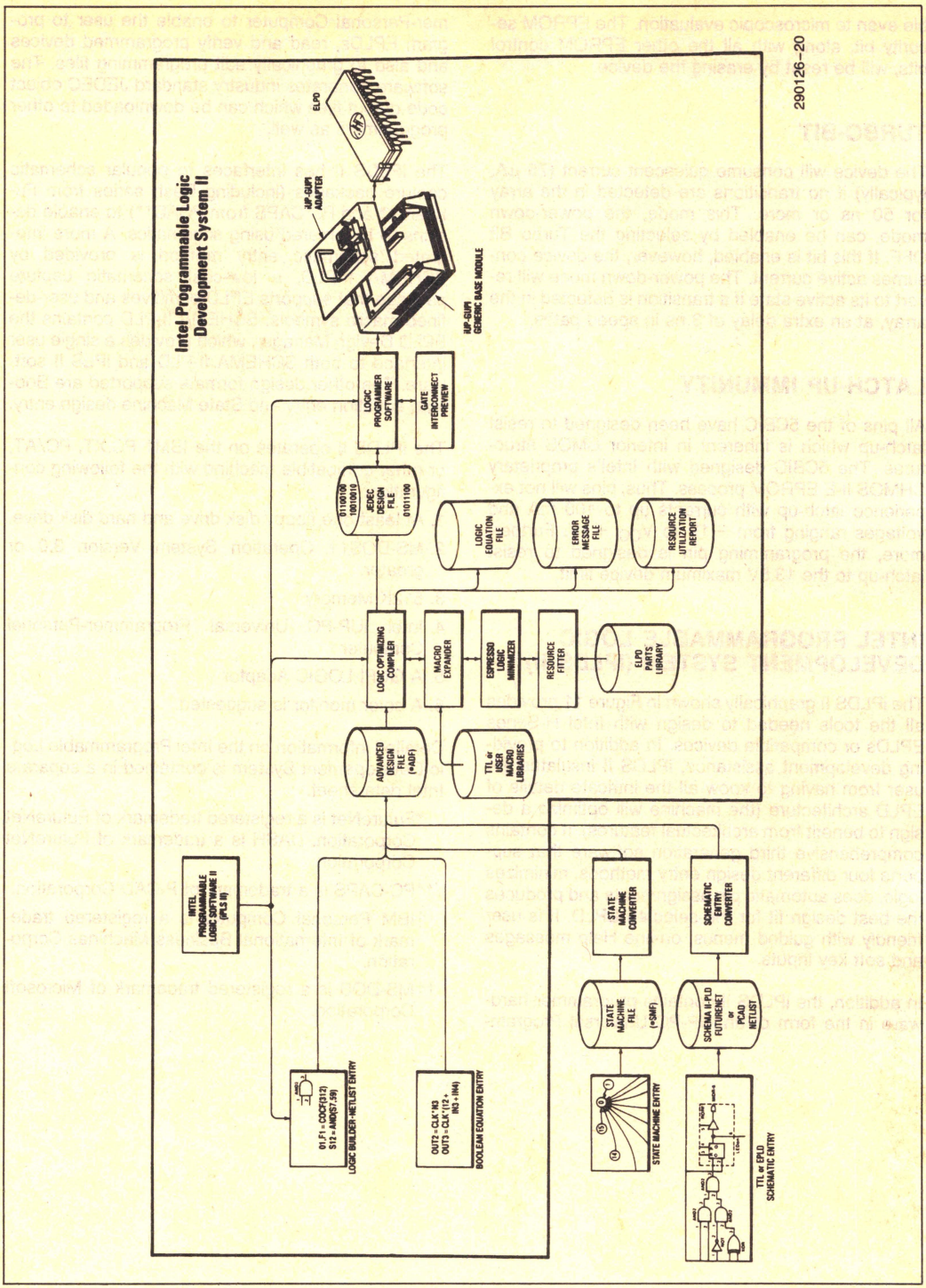


Figure 11. iPLDS II Intel Programmable Logic Development System

intel

APPLICATION NOTE

AP-305

October 1987

Dual-Port Memory Control Using The 5CBIC

NAGEEN SHARMA
PROGRAMMABLE LOGIC APPLICATIONS
INTEL CORPORATION

Order Number: 292032-002

INTRODUCTION

One of the popular multi-port configurations commonly used in multiple microprocessor systems is the dual port. Because each processor is now capable of handling separate tasks in parallel, such designs offer improved performance and throughput. Sharing resources, such as large amounts of memory, is an optimizing trade-off to keep the system efficient and, at the same time, cost-effective.

The scheme discussed here consists of two processors sharing memory through some intermediate logic. Typically, such logic consists of data transceivers, address latches, SSI/MSI arbitration logic (AND gates, OR gates, FLIP-FLOPS etc.). With the 5CBIC implementation, it is possible to reduce the overall chip count by over three times.

A block diagram of the dual-port scheme is shown in Figure 1 for two sixteen-bit processors accessing shared memory. Two 5CBIC's are required for the implementation as all ports in the chip are byte-wide. The first device provides the isolation of the memory and processors' high-byte data bus; it also includes the necessary arbitration logic. The second 5CBIC interfaces the low-byte data bus and implements a 7-bit counter with a parallel-load capability. The lead configurations of the devices are given in Figures 2 and 3.

A bus-arbitration flow-diagram and the state machine diagram for the arbiter are shown in Figures 4 and 5, respectively. These diagrams are translated into equations, which are shown in Figures 6 and 7.

DESCRIPTION

The block diagram (Figure 1) shows two processors, Processor 1 and Processor 2, in a minimal memory system. The interface logic can be condensed to two 5CBIC's. These devices provide the necessary isolation of the shared memory data bus (MDATA [0 . . 15]) from each of the processors' data bus (P1DATA [0 . . 15] and P2DATA [0 . . 15]).

Similar isolation is required for the address bus. This is implemented by a set of latches. It is interesting to note that these latches can also be easily configured in an extra 5CBIC. An implementation of latches can be found in another application note that serves as a multiplexed address/data interface.

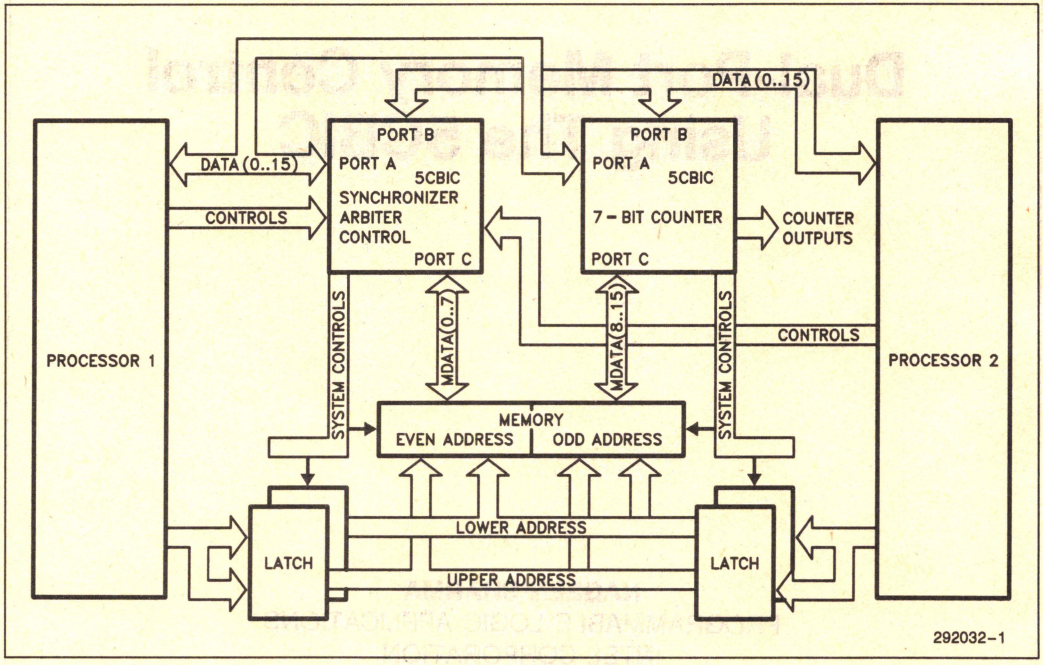


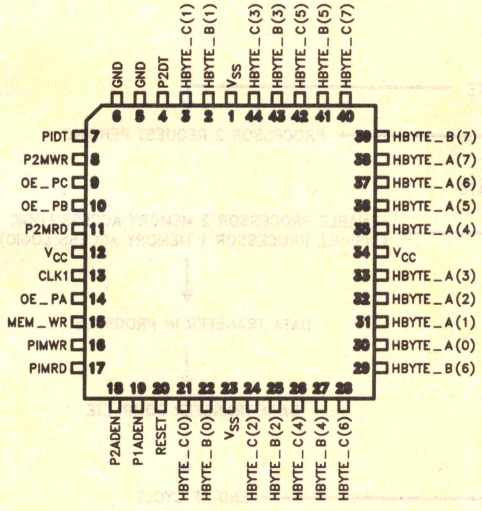
Figure 1. The Dual-Port Block Diagram

The control bus provides all the necessary signals that are used to initiate requests (P1MRD, P1MWR, P2MWR), or simply to provide handshaking signals indicating cycle termination etc.

The arbitration scheme is shown in Figure 4 with the help of a flow-chart. In this example, Processor 1 has been assigned higher priority than Processor 2 to prevent contention. If Processor 2 requests an access while Processor 1 has control of the system bus, the Processor 2 bus cycle is extended by inserting wait states. The cycle remains extended until the arbiter grants access to Processor 2 and enables the appropriate port (PORT B) of the Bus Management Unit (BMU) in the 5CBIC. After the data transfer has occurred, an acknowledge signal signifies cycle completion.

The state machine diagram of the arbitration algorithm is shown in Figure 5. Upon receiving a request from Processor 1, the arbiter will move from it's IDLE state to GRANT1 state and cycle back only after completing the memory access. If Processor 2 requests for a memory access while the arbiter is in the IDLE state, transition to GRANT2 state will occur only if Processor 1 is not requesting access. Different arbitration algorithms are possible; these would translate into different state-machines.

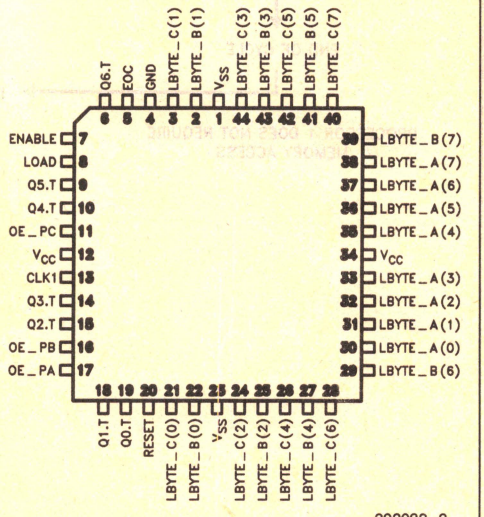
High Byte Data Transfer and Arbitration Logic



292032-2

Figure 2. Lead Configuration

Low Byte Data Transfer and Seven-Bit Counter



292032-3

Figure 3. Lead Configuration

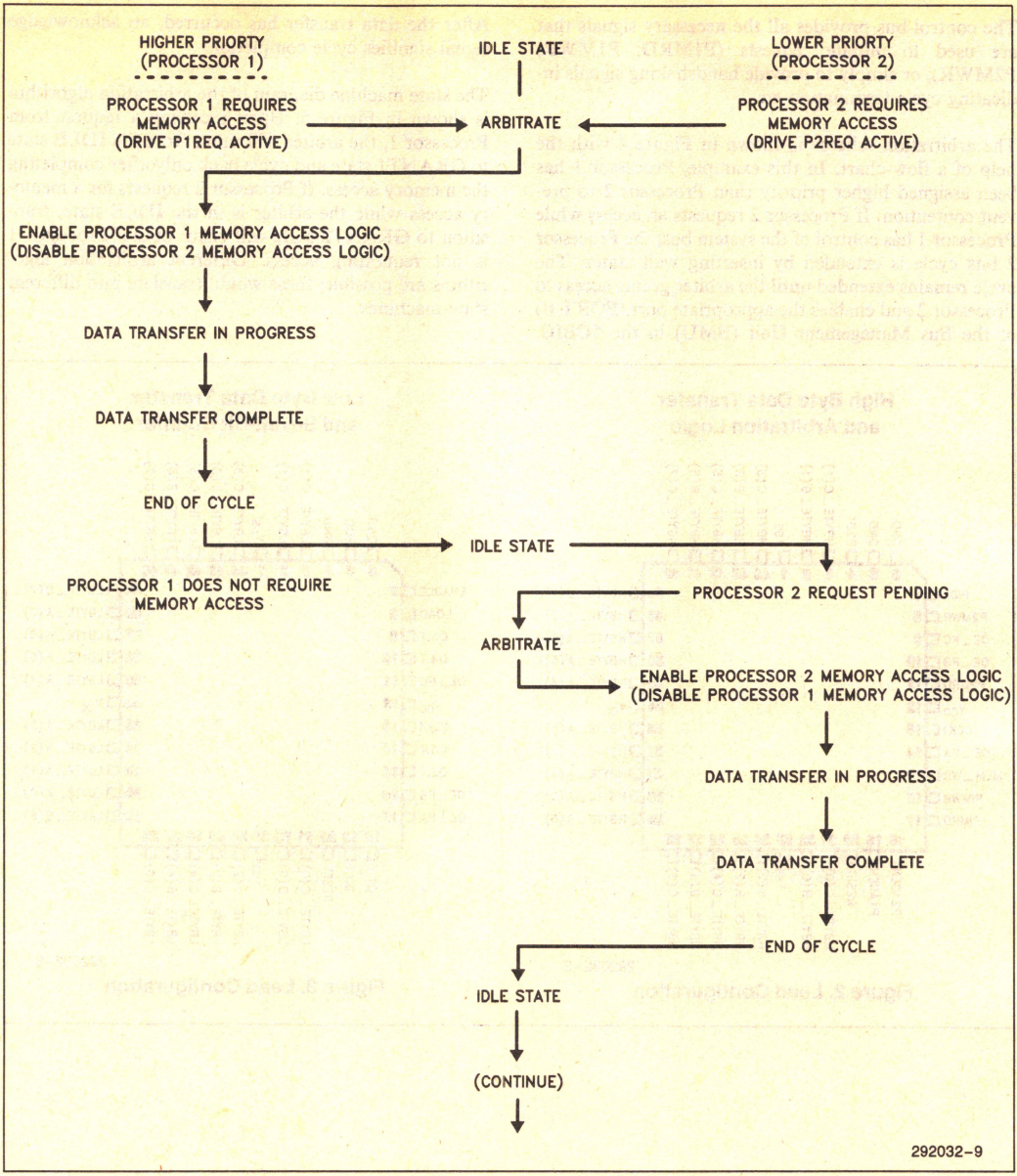


Figure 4. Bus Arbitration Flow Diagram

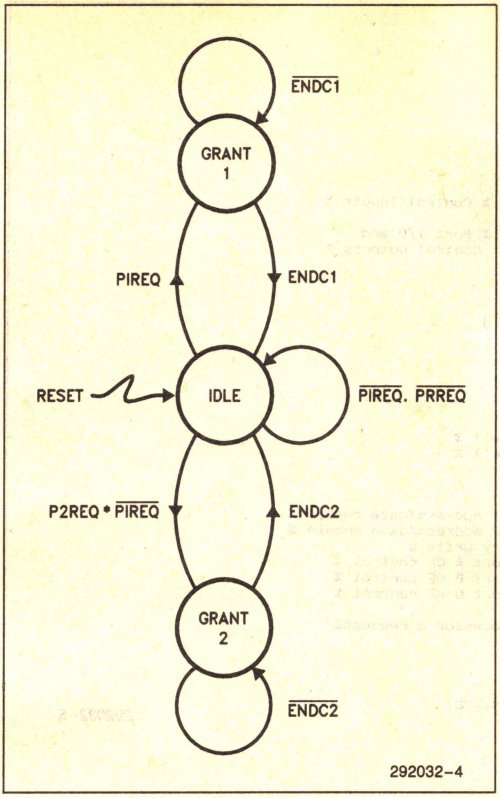


Figure 5. State Diagram

States	State Variables	
	P1GNT	P2GNT
IDLE	0	0
GRANT1	1	0
GRANT2	0	1
ILLEGAL	1	1

It should be pointed out that since Processor 2 could be running on a different clock than Processor 1, it may be necessary to synchronize the requests using the system clock (say Processsor 1). This is conveniently done using buried master/slave flip-flops to prevent erroneous requests or noise from triggering the arbiter states.

The outputs from the arbiter (MEM__WR, P1ADEN, P2ADEN) control the cycle type (READ/WRITE) and the control signals that provide address and data isolation. For the data path, these are generated in the Programmable Logic Unit (PLU) and internally routed to the Bus Management Unit (BMU). This saves board space that is normally occupied by interconnecting traces.

The Bus Management Unit (BMU) is easily configured using the iPLS II Device Configuration Module (DCM) in a high-level graphic fashion. The logic for the arbiter or the 7-bit counter can be entered using the Logic Builder. It should be noted that the 7-bit counter with parallel load is implemented using buried registers with an output EOC (end of count), indicating the completion of the desired count. The load operation is performed using PORT A fed internally to the programmable EPROM array.

```
N. SHARMA
INTEL CORP.
NOVEMBER 10, 1986
1
A
5CB1C
Dual-Port.ADF

OPTIONS: TURBO=OFF
PART: 5CB1C
INPUTS: RESET P1MRD P1MWR P2MRD P2MWR P1DT P2DT CLK1   % Control Inputs %
OUTPUTS: P1ADEN P2ADEN MEM_WR OE_PA OE_PB OE_PC
         HBYTE_A  HBYTE_B  HBYTE_C                    % Port I/O and
                                                        Control outputs %

NETWORK:

%INPUTS%
CLK = INP(CLK1)          % System clock %
RESET = INP(RESET)       % Resest input %
P1MRD = INP(P1MRD)       % Proc. 1 memory read %
P1MWR = INP(P1MWR)       % Proc. 1 memory write %
P2MRD = INP(P2MRD)       % Proc. 2 memory read %
P2MWR = INP(P2MWR)       % Proc. 2 memory write %
P1DT = INP(P1DT)         % Proc. 1 data (transmit/receive') %
P2DT = INP(P2DT)         % Proc. 2 data (transmit/receive') %

%OUTPUTS%
P1ADEN,P1ADEN = RORF (P1GNT,CLK,GND,GND,VCC)  % Proc. 1 address/data enable %
P2ADEN,P2ADEN = RORF (P2GNT,CLK,GND,GND,VCC)  % Proc. 2 address/data enable %
MEM_WR = CONF (MEM_WR,VCC)              % Dual-Port memory write %
OE_PA,OEPA = COCF (OE_PA,VCC)            % Control for Port A OE control %
OE_PB,OEPB = COCF (OE_PB,VCC)            % Control for Port B OE control %
OE_PC,OEPC = COCF (OE_PC,VCC)            % Control for Port C OE control %

P2REQ = NORF (SYN_P2RQ,CLK,GND,GND)  % Synchronized processor 2 request%

% PORTS IN BMU %

HBYTE_A,HBYTE_B,HBYTE_C = BMU (Ob110000,OEPA,GND,VCC,
                               OEPB,GND,VCC,OEPC,P1ADEN,VCC)
```

292032-5

```
EQUATIONS:

  % ARBITRATION %
P1GNT = /RESET * (P1REQ * /P1ADEN * /P2ADEN +
                  /P2ADEN * P1ADEN * /ENDC1 )   ; % Proc. 1 granted access %

P2GNT = /RESET * (SYN_P2RQ * /P1REQ * /P2ADEN * /P1ADEN +
                  P2ADEN * /P1ADEN * /ENDC2 )    ; % Proc. 2 granted access %

% CONTROL %

P1REQ = P1MRD + P1MWR ;          % Proc. 1 request %
SYN_P2RQ = P2MRD + P2MWR ;         % Proc. 2 request %
ENDC1 = /P1REQ          ;        % Proc. 1 end of cycle %
ENDC2 = /SYN_P2RQ       ;          % Proc. 2 end of cycle %
MEM_WR = P1MWR * P1ADEN +
         P2MWR * P2ADEN;         % Memory write %
OE_PA = P1ADEN * /P1DT ;          % Control signal for Port A %
OE_PB = P2ADEN * /P2DT ;          % Control signal for Port B %
OE_PC = P1ADEN * P1DT + P2ADEN * P2DT ; % Control signal for Port  C %

END$
```

292032-6

Figure 6. ADF Listing for Arbiter and High Byte Data Transfer Logic

```
N. SHARMA
INTEL CORP.
AUGUST 31, 1987
2
A
5CBIC
Dual-Port( Low byte ).ADF AND AN ADDRESSABLE SEVEN-BIT DOWN COUNTER

OPTIONS: TURBO=OFF
PART: 5CBIC
INPUTS:  CLK1  RESET  OE_PA  OE_PB  OE_PC  LOAD  ENABLE P1ADEN
                              % Control Inputs %
OUTPUTS:  LBYTE_A  LBYTE_B  LBYTE_C
          Q0.T  Q1.T  Q2.T  Q3.T  Q4.T  Q5.T  Q6.T  EOC
                            % BMU and counter outputs  %


NETWORK:

%INPUTS%
CLK = INP(CLK1)          % System clock %
OE_PA = INP(OE_PA)
OE_PB = INP(OE_PB)
OE_PC = INP(OE_PC)
LOAD = INP(LOAD)
CLEAR = INP(RESET)
P1ADEN = INP (P1ADEN)
P2ADEN = INP (P2ADEN)
ENABLE = INP (ENABLE)


%OUTPUTS%
Q0.T,Q0 = TOTF ( Q0.T,CLK,CLEAR,GND,VCC )        % COUNTER STATES %
Q1.T,Q1 = TOTF ( Q1.T,CLK,CLEAR,GND,VCC )
Q2.T,Q2 = TOTF ( Q2.T,CLK,CLEAR,GND,VCC )
Q3.T,Q3 = TOTF ( Q3.T,CLK,CLEAR,GND,VCC )
Q4.T,Q4 = TOTF ( Q4.T,CLK,CLEAR,GND,VCC )
Q5.T,Q5 = TOTF ( Q5.T,CLK,CLEAR,GND,VCC )
Q6.T,Q6 = TOTF ( Q6.T,CLK,CLEAR,GND,VCC )
EOC = CONF(EOCT,VCC)    % OUTPUT FOR SYSTEM %

% PORTS IN BMU %

LBYTE_A,LBYTE_B, LBYTE_C = BMU (0b110000,OE_PA,GND,VCC,
                                OE_PB,GND,VCC,OE_PC,P1ADEN,VCC)

DATA[0:7] = BFMX (GND,GND)  % BYTE FOR INITIALIZING COUNTER %                    292032-7
EQUATIONS:

Q0.T = (LOAD * (DATA[0] * /Q0) + (/DATA[0] * Q0) ) +      % LOAD OPERATION %
       (/LOAD * ENABLE )      ;                          % COUNT %

Q1.T = (LOAD * (DATA[1] * /Q1) + (/DATA[1] * Q1) ) +
       /LOAD * ENABLE *  /Q0 ) ;

Q2.T = (LOAD * (DATA[2] * /Q2) + (/DATA[2] * Q2) ) +
       (/LOAD * ENABLE *  /Q1 * /Q0) ;

Q3.T = (LOAD * (DATA[3] * /Q3) + (/DATA[3] * Q3) ) +
       (/LOAD * ENABLE * /Q2 * /Q1 * /Q0) ;

Q4.T = (LOAD * (DATA[4] * /Q4) + (/DATA[4] * Q4) ) +
       (/LOAD * ENABLE * /Q3 * /Q2 * /Q1 * /Q0 ) ;

Q5.T = (LOAD * (DATA[5] * /Q5) + (/DATA[5] * Q5) ) +
       (/LOAD * ENABLE * /Q4 * /Q3 * /Q2 * /Q1 * /Q0 ) ;

Q6.T = (LOAD * (DATA[6] * /Q6) + (/DATA[6] * Q6) ) +
       (/LOAD * ENABLE * /Q5 * /Q4 * /Q3 * /Q2 * /Q1 * /Q0 ) ;

EOCT = /Q6 * /Q5 * /Q4 * /Q3 * /Q2 * /Q1 * /Q0  ;


END$
```

292032-8

Figure 7. ADF Listing for Low Byte Data Transfer Logic and Seven-Bit Addressable Counter

intel

APPLICATION NOTE

AP-308

October 1987

The Multiplexed Bus Interface with the 5CBIC

NAGEEN SHARMA
PROGRAMMABLE LOGIC APPLICATIONS
INTEL CORPORATION

Order Number: 292035-002

INTRODUCTION

When designing with microprocessors or other LSI chips that employ time-multiplexed buses, the demultiplexing logic typically consists of latches and transceivers. Further, decoding is accomplished using extra chips to generate chip selects. It is also common to need a wait-state generator to accommodate devices with different access times. Other specialized functions, like a barrel-shifter, are also frequently needed to shift data in data-processing applications and floating point arithmetic.

The Intel 5CBIC EPLD has the logic density and architectural flexibility to combine all these functions in a single device. This integration results in a three-fold savings in device count over an SSI/MSI approach for a typical multiplexed bus interface used in 16-bit microprocessor systems.

A block diagram of the system is shown in Figure 1 with the lead configurations of the two 5CBIC's in Figures 2 and 3. The address map for the decoder and the state-table for the wait-state generator are shown in Figures 4 and 5, respectively. The truth table for the barrel-shifter is given in Figure 6 and the block diagram in Figure 7. The equations implementing the high-byte demultiplexing of the address/data bus, address decoding and generating the appropriate number of wait-states, are listed in the .adf (advanced design file) format in Figure 8. The equations for the corresponding low-byte demultiplexing logic and an eight bit barrel-shifter are listed in Figure 9.

DESCRIPTION

Since the 5CBIC is a byte-oriented device, two devices are required when interfacing to a 16-bit microprocessor or microcontroller. The address/data bus from the processor is connected to Port A of the 5CBIC, with the demultiplexed data bus and address bus on the high drive Ports B and C, respectively. The logic between Port A and Port B consists of a transceiver; the logic between Ports A and C is configured as a transparent latch.

The high-order address bits are internally fed from the Bus Management Unit (BMU) to the Programmable Logic Unit (PLU) for decoding. This example uses a microprocessor with a 20-bit address bus, and therefore the high-address lines A17 . . . A19 can be latched in the PLU input macrocells. This scheme can be configured using the Device Configuration Manager (DCM) in a high-level graphic fashion, available in the iPLDS-II (Intel Programmable Logic Development System).

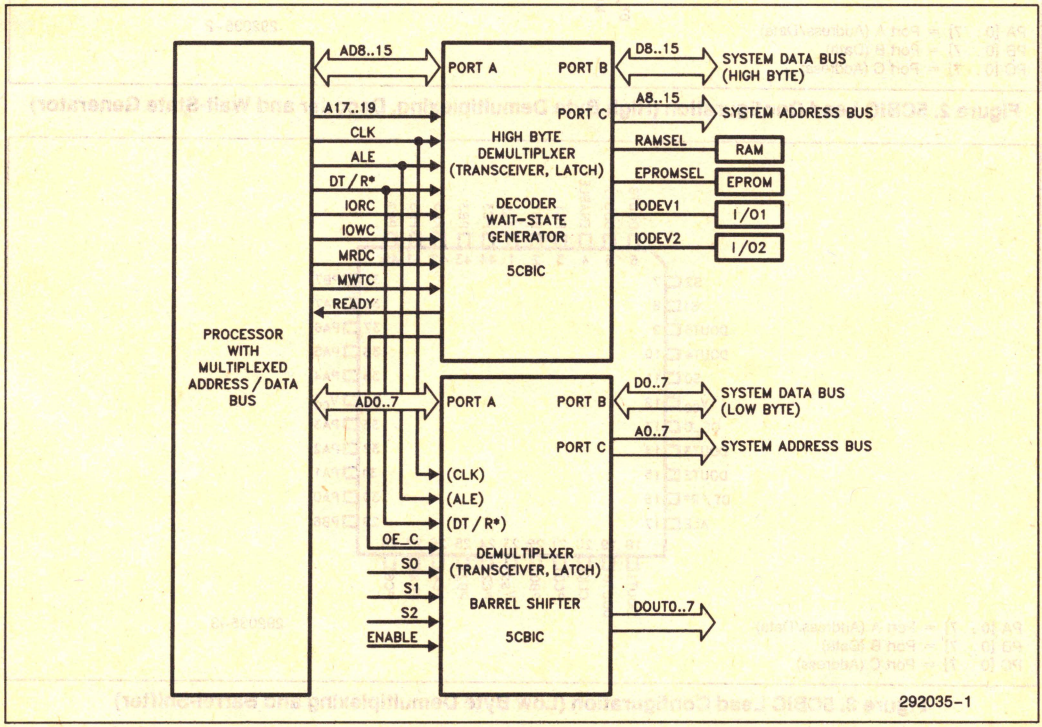


Figure 1. Block Diagram of a Minimal System Using Multiplexed Address/Data Bus

Decoding is accomplished using the latched addresses and the appropriate control signals. Along with the chip-selects, buried logic implements a wait-state generator to introduce up to six extra states in the processor cycle. (Buried logic means that the macrocells in the PLU can be used independently of their input pins). Figure 5 shows the table for a down counter with load capability and the corresponding number of wait-states for the devices in the system.

The lower address/data bus lines are demultiplexed using the second 5CBIC, the equations for this device are listed in Figure 9. The PLU here is used to implement an eight-bit barrel-shifter, that shifts the data (loaded through the BMU, Port B) a fixed number of bits based on status lines SO . . S2. The outputs are made available on the I/O port in the PLU.

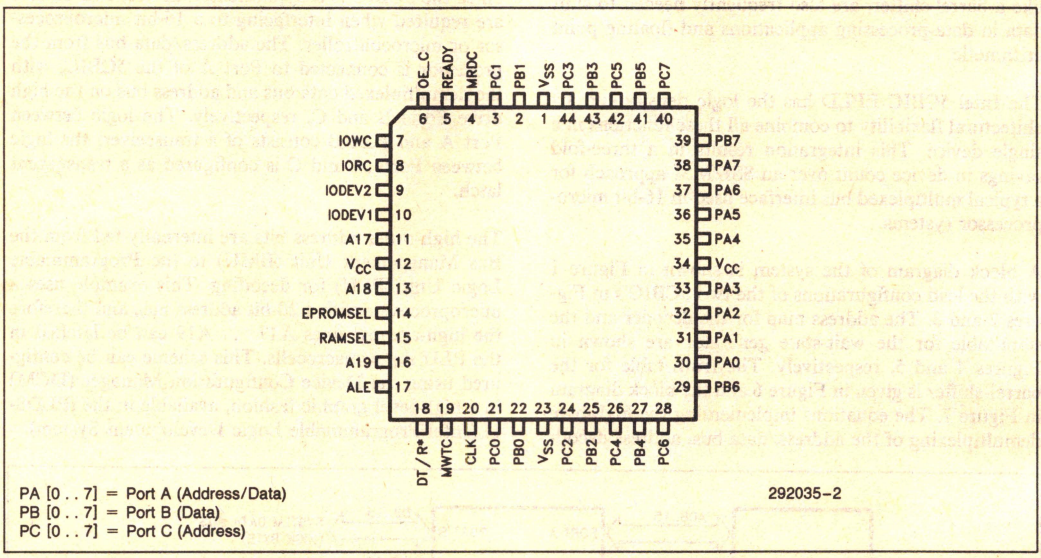


Figure 2. 5CBIC Lead Configuration (High Byte Demultiplexing, Decoder and Wait-State Generator)

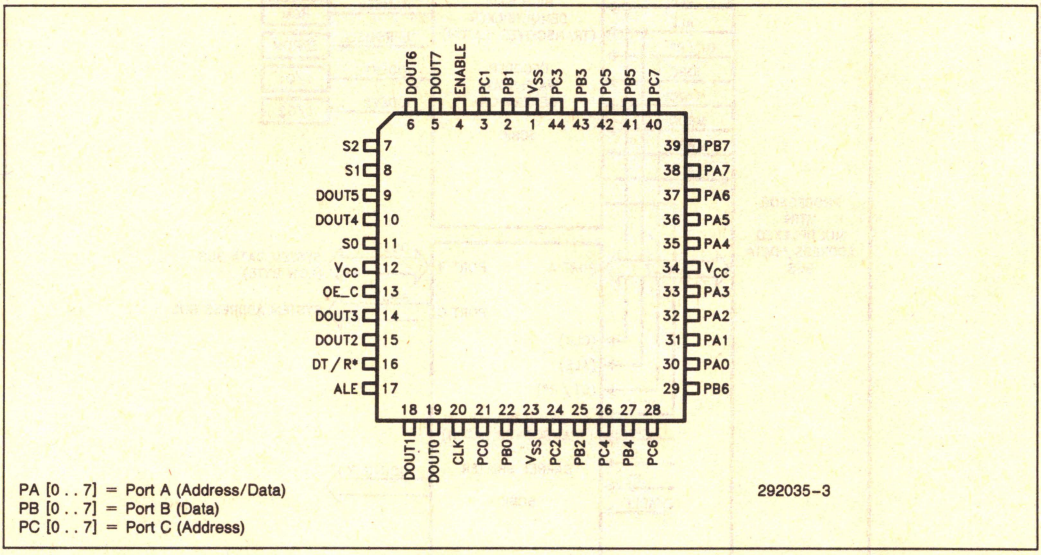


Figure 3. 5CBIC Lead Configuration (Low Byte Demultiplexing and Barrel-Shifter)

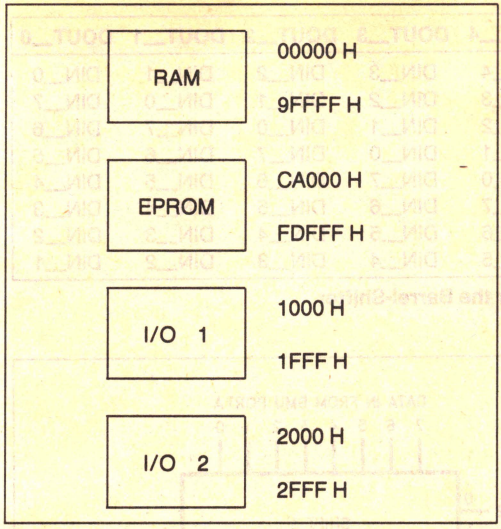


Figure 4. Address Map

The READY signal is used to increase cycle time for devices that cannot transfer data at maximum processor bus bandwidth.

This implementation illustrates not only a path of higher integration in this very common design, but also the benefits that can be realized if the system is viewed as a collection of smaller functions. These can then be paired together to best exploit the full capability of the 5CBIC.

The amount of shift in data output is determined by the select lines S0, S1 and S2. This can be used for formatting data needed in other sections in a system.

State Variables			# Wait States	Device
Q2	Q1	Q0		
1	1	1	7	
1	1	0	6	IODEV2
1	0	1	5	
1	0	0	4	
0	1	1	3	
0	1	0	2	IODEV1
0	0	1	1	EPROM
0	0	0	0	ENABLE READY

Figure 5a. State Table for the Wait-State Generator

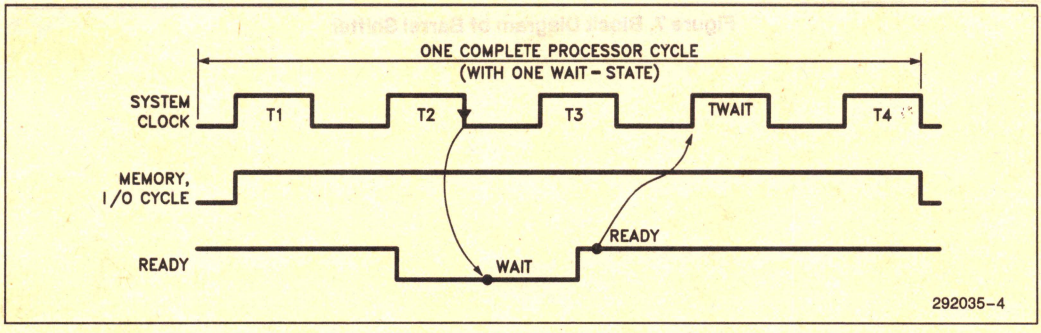


Figure 5b. Waveforms Showing Wait State Insertion in Processor Cycle

S2	S1	SO	DOUT_7	DOUT_6	DOUT_5	DOUT_4	DOUT_3	DOUT_2	DOUT_1	DOUT_0
0	0	0	DIN_7	DIN_6	DIN_5	DIN_4	DIN_3	DIN_2	DIN_1	DIN_0
0	0	1	DIN_6	DIN_5	DIN_4	DIN_3	DIN_2	DIN_1	DIN_0	DIN_7
0	1	0	DIN_5	DIN_4	DIN_3	DIN_2	DIN_1	DIN_0	DIN_7	DIN_6
0	1	1	DIN_4	DIN_3	DIN_2	DIN_1	DIN_0	DIN_7	DIN_6	DIN_5
1	0	0	DIN_3	DIN_2	DIN_1	DIN_0	DIN_7	DIN_6	DIN_5	DIN_4
1	0	1	DIN_2	DIN_1	DIN_0	DIN_7	DIN_6	DIN_5	DIN_4	DIN_3
1	1	0	DIN_1	DIN_0	DIN_7	DIN_6	DIN_5	DIN_4	DIN_3	DIN_2
1	1	1	DIN_0	DIN_7	DIN_6	DIN_5	DIN_4	DIN_3	DIN_2	DIN_1

Figure 6. Truth-Table for the Barrel-Shifter

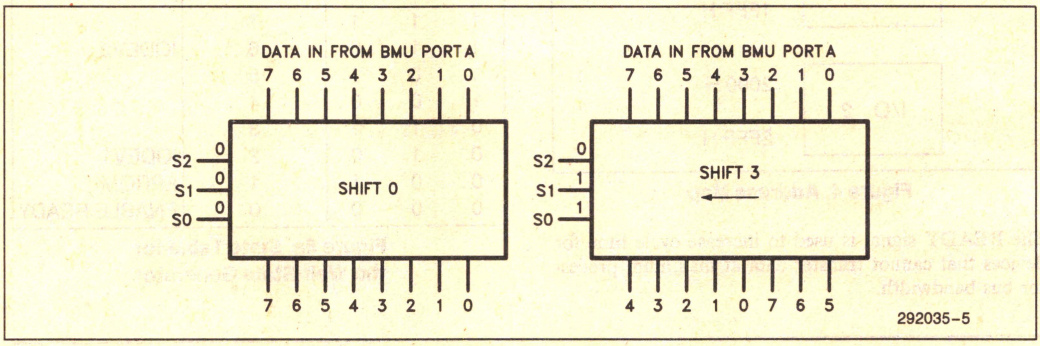


Figure 7. Block Diagram of Barrel Shifter

```
N. SHARMA
INTEL CORP.
AUGUST 31, 1987
2
A
5CBIC
MUL.ADF -- DEMULTIPLEXER, DECODER, WAIT-STATE GENERATOR

OPTIONS: TURBO=OFF
PART: 5CBIC
INPUTS:  CLK  ALE  A19  A18  A17  IORC  IOWC  MRDC  MWTC  DT/R*
OUTPUTS: PA_ADDT  PB_DT  PC_AD  RAMSEL  EPROMSEL  IODEV1  IODEV2  OE_C  READY

NETWORK:
CLK = INP (CLK)
A19 = LINP (A19)
A18 = LINP (A18)
A17 = LINP (A17)
ALE = INP (ALE)
IORC = INP (IORC)
IOWC = INP (IOWC)
MRDC = INP (MRDC)
MWTC = INP (MWTC)
DTR  = INP (DT/R*)


PA_ADDT,PB_DT,PC_AD = BMU (0b111011,OE_A,VCC,ALE,OE_B,GND,VCC,OE_C,GND,VCC)
LA[0:7] = BFMX (VCC,GND)       % LATCHED ADDRESSES 8..15 FED BACK INTO %
                                % THE PLU FOR I/O DECODING                %
OE_C,OE_C = COCF(OEC,VCC)       % OUTPUT CONTROL FOR PORT C %

RAMSEL = CONF (RAMSL,VCC)                    % RAM SELECT %
EPROMSEL,EPROMSEL = COCF (EPROMSL,VCC)         % EPROM SELECT %
IODEV1,IODEV1 = COCF (IODV1,VCC)             % I/O DEVICE 1 SELECT %
IODEV2,IODEV2 = COCF (IODV2,VCC)             % I/O DEVICE 2 SELECT %


READY = CONF (RDY,RDYOE)              % READY FOR CYCLE COMPLETION %

Q0 = NORF (Q0D,CLK,GND,GND)          % STATES Q0..Q2 FOR WAIT-STATE %
Q1 = NORF (Q1D,CLK,GND,GND)          % GENERATION ( USING BURIED    %
Q2 = NORF (Q2D,CLK,GND,GND)          %  REGISTERS )                 %
```

292035-6

```
EQUATIONS:

% DECODE EQUATIONS %

RAMSL = /A18 * /A17 * MREQ ;                  % 00000H - 9FFFFH %
EPROMSL = A19 * A18 * LA[7] * MREQ ;           % CA000H - FDFFFH %
IODV1 = /LA[7] * /LA[6] * /LA[5] * LA[4] * IOREQ ;    %  1000H - 1FFFH  %
IODV2 = /LA[7] * /LA[6] * LA[5] * /LA[4] * IOREQ ;    %  2000H - 2FFFH  %

MREQ = (MRDC + MWTC) ;              % INTERMEDIATE EQUATIONS FOR MEMORY %
IOREQ = (IORC + IOWC);              % AND I/O REQUESTS %
OE_A = DTR  ;
OE_B = /DTR   ;
OEC = MRDC + MWTC + IORC + IOWC ;


%  THE FOLLOWING IS A WAIT-STATE GENERATOR FOR MEMORY AND I/O REQUESTS %

Q0D = /Q0 * /ALE
      + EPROMSEL * ALE ;


Q1D = Q1 * /Q0 * /ALE
      + /Q1 * /Q0 * /ALE
      + IODEV1 * ALE
      + IODEV2 * ALE ;

Q2D = Q2 * /Q1 * /Q0 * /ALE
      + /Q2 * /Q1 * /Q0 * /ALE
      + IODEV2 * ALE   ;

RDY = GND ;
RDYOE = /Q0 * /Q1 * /Q2  ;



END$
```

292035-7

Figure 8. ADF Listing for Demultiplexing Higher Address/Data Byte, Decoder, Wait-State Generator

```
N. SHARMA
INTEL CORP.
AUGUST 31, 1987
2
A
5CBIC
MULH.ADF -- DEMULTIPLEXER, BARREL SHIFTER

OPTIONS: TURBO=OFF
PART: 5CBIC
INPUTS:  CLK  ALE  DT/R*  OE_C  S0  S1  S2  ENABLE
OUTPUTS: PA_ADDTL  PB_DTL  PC_ADL  DOUT0  DOUT1  DOUT2  DOUT3  DOUT4  DOUT5
         DOUT6 DOUT7
         % PA_ADDTL = ADDRESS/DATA BUS ON PORT A (LOW BYTE) %
         % PB_DTL = SYSTEM DATA BUS ON PORT B (LOW BYTE )   %
         % PC_ADL = SYSTEM ADDRESS BUS ON PORT C (LOW BYTE) %
         % DOUT0..7 = SHIFTED DATA OUTPUT ON I/O PORT       %

NETWORK:
CLK = INP (CLK)
ALE = INP (ALE)
DTR  = INP (DT/R*)
OE_C = INP (OE_C)
S0 = INP (S0)
S1 = INP (S1)
S2 = INP (S2)
ENABLE = INP (ENABLE)

PA_ADDTL,PB_DTL,PC_ADL = BMU (0b111011,OE_A,VCC,ALE,OE_B,GND,VCC,OE_C,GND,VCC)

D_IN[0:7] = BFMUX (VCC,GND)        % DATA LOADED INTO     %
                                   % THE PLU FOR SHIFTING %
                                   % FROM PORT A          %
DOUT0 =  RONF (DO0,CLK,GND,GND,ENABLE)     % DATA OUTPUT %
DOUT1 = RONF (DO1,CLK,GND,GND,ENABLE)
DOUT2 = RONF (DO2,CLK,GND,GND,ENABLE)
DOUT3 = RONF (DO3,CLK,GND,GND,ENABLE)
DOUT4 = RONF (DO4,CLK,GND,GND,ENABLE)
DOUT5 = RONF (DO5,CLK,GND,GND,ENABLE)
DOUT6 = RONF (DO6,CLK,GND,GND,ENABLE)
DOUT7 = RONF (DO7,CLK,GND,GND,ENABLE)                                  292035-8

EQUATIONS:

% CONTROL INPUTS TO THE BMU %

OE_A = DTR  ;
OE_B = /DTR ;
% OE_C IS AN INPUT FROM THE HIGHER BYTE DEMULTIPLEXING 5CBIC %

% THE FOLLOWING IMPLEMENTS A BARREL SHIFTER %
SHIFT0 = /S2 * /S1 * /S0 ;  % INTERMEDIATE SHIFT EQUATIONS %
SHIFT1 = /S2 * /S1 * S0 ;
SHIFT2 = /S2 * S1 * /S0 ;
SHIFT3 = /S2 * S1 * S0 ;
SHIFT4 = S2 * /S1 * /S0 ;
SHIFT5 = S2 * /S1 * S0 ;
SHIFT6 = S2 * S1 * /S0 ;
SHIFT7 = S2 * S1 * S0 ;

DO0 = SHIFT0 * D_IN[0]
    + SHIFT1 * D_IN[7]
    + SHIFT2 * D_IN[6]
    + SHIFT3 * D_IN[5]
    + SHIFT4 * D_IN[4]
    + SHIFT5 * D_IN[3]
    + SHIFT6 * D_IN[2]
    + SHIFT7 * D_IN[1]   ;

DO1 = SHIFT0 * D_IN[1]
    + SHIFT1 * D_IN[0]
    + SHIFT2 * D_IN[7]
    + SHIFT3 * D_IN[6]
    + SHIFT4 * D_IN[5]
    + SHIFT5 * D_IN[4]
    + SHIFT6 * D_IN[3]
    + SHIFT7 * D_IN[2]   ;

DO2 = SHIFT0 * D_IN[2]
    + SHIFT1 * D_IN[1]
    + SHIFT2 * D_IN[0]
    + SHIFT3 * D_IN[7]
    + SHIFT4 * D_IN[6]
    + SHIFT5 * D_IN[5]
    + SHIFT6 * D_IN[4]
    + SHIFT7 * D_IN[3]   ;
                                                                       292035-9
```

Figure 9. ADF Listing for Demultiplexing Lower Address/Data Byte and Barrel-Shifter

```
DO3 = SHIFT0 * D_IN[3]
    + SHIFT1 * D_IN[2]
    + SHIFT2 * D_IN[1]
    + SHIFT3 * D_IN[0]
    + SHIFT4 * D_IN[7]
    + SHIFT5 * D_IN[6]
    + SHIFT6 * D_IN[5]
    + SHIFT7 * D_IN[4]   ;

DO4 = SHIFT0 * D_IN[4]
    + SHIFT1 * D_IN[3]
    + SHIFT2 * D_IN[2]
    + SHIFT3 * D_IN[1]
    + SHIFT4 * D_IN[0]
    + SHIFT5 * D_IN[7]
    + SHIFT6 * D_IN[6]
    + SHIFT7 * D_IN[5]   ;

DO5 = SHIFT0 * D_IN[5]
    + SHIFT1 * D_IN[4]
    + SHIFT2 * D_IN[3]
    + SHIFT3 * D_IN[2]
    + SHIFT4 * D_IN[1]
    + SHIFT5 * D_IN[0]
    + SHIFT6 * D_IN[7]
    + SHIFT7 * D_IN[6]   ;

DO6 = SHIFT0 * D_IN[6]
    + SHIFT1 * D_IN[5]
    + SHIFT2 * D_IN[4]
    + SHIFT3 * D_IN[3]
    + SHIFT4 * D_IN[2]
    + SHIFT5 * D_IN[1]
    + SHIFT6 * D_IN[0]
    + SHIFT7 * D_IN[7]   ;

DO7 = SHIFT0 * D_IN[7]
    + SHIFT1 * D_IN[6]
    + SHIFT2 * D_IN[5]
    + SHIFT3 * D_IN[4]
    + SHIFT4 * D_IN[3]
    + SHIFT5 * D_IN[2]
    + SHIFT6 * D_IN[1]
    + SHIFT7 * D_IN[0]   ;

END$
```

292035-10

Figure 9. ADF Listing for Demultiplexing Lower Address/Data Byte and Barrel-Shifter (Continued)

intel

APPLICATION NOTE

AP-309

January 1987

DRAM Address Interface with the 5CBIC

NAGEEN SHARMA
PROGRAMMABLE LOGIC APPLICATIONS

Order Number: 292036-001

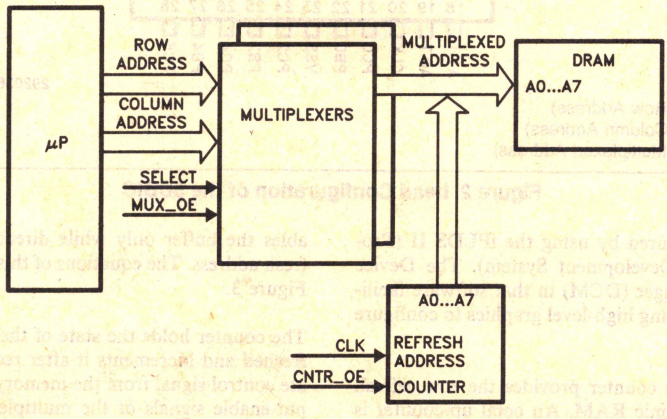
INTRODUCTION

In most DRAM applications, the row and the column addresses for data transfers must be multiplexed on a common bus. Refresh circuitry, however, provides the refresh address at fixed intervals. This note describes an implementation of both the multiplexing circuitry and the refresh address logic in a one-chip replacement of the two multiplexers and one counter needed in a conventional SSI/MSI design.

The block diagram of the DRAM address interface is shown in Figure 1. The lead configuration of the 5CBIC is shown in Figure 2. The equations for the design are provided in Figure 3 in the advanced design file (ADF).

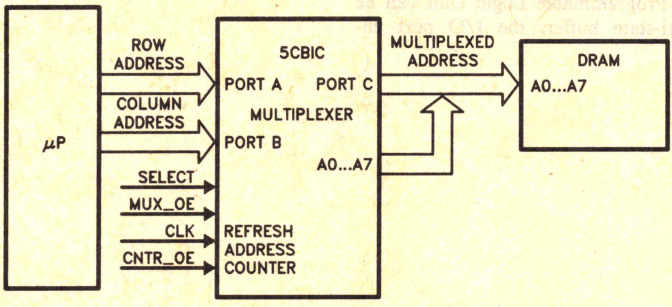
DESCRIPTION

The block diagram shown in Figure 1 is the address interface needed in dynamic RAM circuits. The port-oriented 5CBIC provides a high integration alternative to discrete SSI/MSI devices such as multiplexers and counters. The row address bus is connected to Port A while the column address is connected to Port B of the Bus Management Unit (BMU). The multiplexed address bus is Port C, a high drive port, that is connected to the DRAM address inputs. The controlling signal for the multiplexing is provided by a memory controller (easily configurable in another EPLD).



292036-1

a. Discrete SSI/MSI Approach (Minimum of Three Devices)



292036-2

b. High Integration 5CBIC Implementation (Single Device)

Figure 1. Block Diagram of DRAM Address Interface

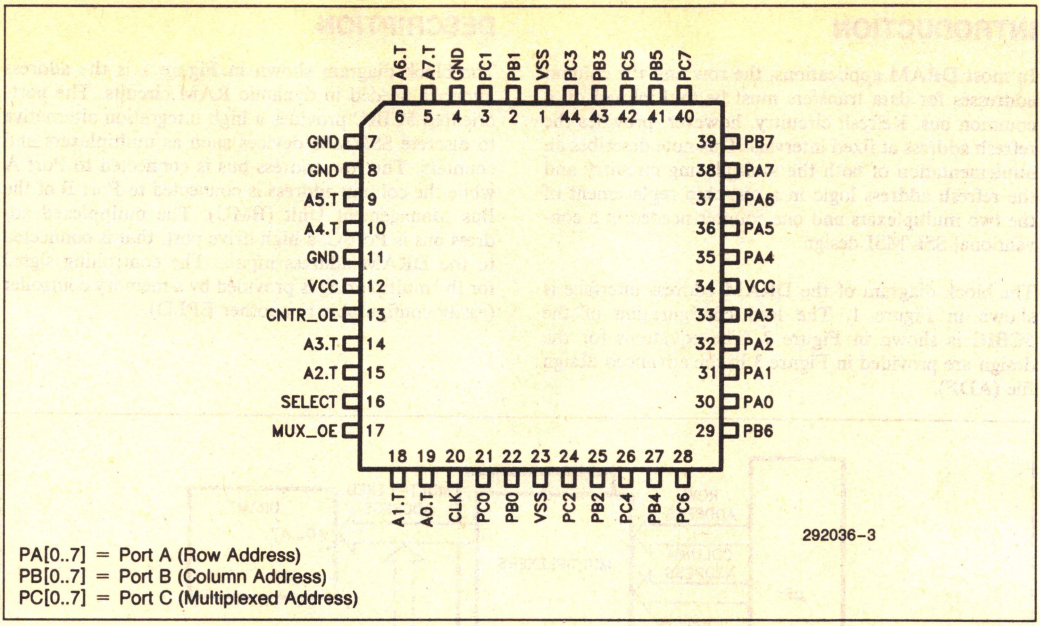


Figure 2. Lead Configuration of the 5CBIC

The BMU is configured by using the iPLDS II (Programmable Logic Development System). The Device Configuration Manager (DCM) in that software facilitates design entry using high-level graphics to configure the BMU.

The refresh address counter provides the row refresh address to the dynamic RAM. An octal up-counter is adequate for most DRAM's (as a number of them require fewer than eight address lines for refresh). Since the outputs of the Programmable Logic Unit can be controlled by a tri-state buffer, the I/O port enables the buffer only while directly providing the refresh address. The equations of this counter are given in Figure 3.

The counter holds the state of the last row address refreshed and increments it after receiving the appropriate control signal from the memory controller. The output-enable signals of the multiplexer and the counter are the only other control signals required for this circuit.

```
N. SHARMA
INTEL CORP.
December 2, 1986
1
A
5CBIC
DRAM Multiplexer and Refresh Address Generator

OPTIONS: TURBO=OFF
PART: 5CBIC
INPUTS:   CLK MUX_OE SELECT CNTR_OE          % Control Inputs %
OUTPUTS:  PA_ROW PB_COL PC_DRAM
          A0.T A1.T A2.T A3.T A4.T A5.T A6.T A7.T
                  % BMU and counter outputs  %
                  % PA_ROW = ROW ADDRESS INPUT AT PORT A %
                  % PB_COL = COLUMN ADDRESS INPUT AT PORT B %
                  % PC_DRAM = MULTIPLEXED ADDRESS OUTPUT FROM %
                  %           PORT C TO DRAM                 %
                  % A0.T .. A7.T = REFRESH ADDRESS COUNTER OUTPUTS %

NETWORK:

%INPUTS%
CLK = INP(CLK)          % CLOCK INPUT%
MUX_OE = INP(MUX_OE)    % MULTIPLEXER OUTPUT ENABLE %
CNTR_OE = INP(CNTR_OE) % COUNTER OUTPUT ENABLE %
SELECT = INP (SELECT)  % MULTIPLEXER OUTPUT SELECT %

%OUTPUTS%
A0.T,A0 = TOTF ( VCC,CLK,GND,GND,CNTR_OE )       % COUNTER STATES %
A1.T,A1 = TOTF ( A1.T,CLK,GND,GND,CNTR_OE )
A2.T,A2 = TOTF ( A2.T,CLK,GND,GND,CNTR_OE )
A3.T,A3 = TOTF ( A3.T,CLK,GND,GND,CNTR_OE )
A4.T,A4 = TOTF ( A4.T,CLK,GND,GND,CNTR_OE )
A5.T,A5 = TOTF ( A5.T,CLK,GND,GND,CNTR_OE )
A6.T,A6 = TOTF ( A6.T,CLK,GND,GND,CNTR_OE )
A7.T = TONF ( A7.T,CLK,GND,GND,CNTR_OE )

% PORTS IN BMU %

PA_ROW,PB_COL,PC_DRAM = BMU (0b000111,GND,GND,VCC,
                             GND,GND,VCC,OE_PC,SEL_PC,GND)
```

292036-4

Figure 3. ADF Listing of the Address Multiplexer and the Refresh Address Counter

```
EQUATIONS:
NMUX_OE = /MUX_OE ;
OE_PC = NMUX_OE ;
SEL_PC = SELECT ;

A1.T = A0  ;

A2.T = A1 * A0 ;

A3.T = A2 * A1 * A0 ;

A4.T = A3 * A2 * A1 * A0  ;

A5.T = A4 * A3 * A2 * A1 * A0  ;

A6.T = A5 * A4 * A3 * A2 * A1 * A0  ;

A7.T = A6 * A5 * A4 * A3 * A2 * A1 * A0  ;

END$
```

292036-5

DESIGN ENTRY

ELECTRONIC DESIGN EXCLUSIVE

Programmable logic shrinks bus-interface designs

Nageen Sharma
Intel Corp., 1900 Prairie City Rd., Folsom, CA 95630; (916) 351-2758.

A CMOS LSI circuit offers high-drive ports for data transfer and programmable controls for bus interfaces. It also cuts power dissipation.

Most microprocessor- or microcontroller-based circuits need external logic to bridge address and data buses to the rest of the system, including memory and serial or parallel inputs and outputs. Designers usually rely on various TTL devices to perform this task. Such assortments of components significantly raise board space and power dissipation. As a result they often require cooling, reduce reliability, and add cost.

Most interfaces, for example, contain bus drivers and transceivers; encoders, decoders and multiplexers; and assorted latches, flip-flops, and counters. Depending on the complexity of the system, these devices make up 70% to 90% of the total chip count and fill almost that percentage of the board space. The advent of dedicated LSI interface chips has eased the congestion somewhat, but many designs still call for a number of SSI and MSI devices.

The 5CBIC bus interface controller brings a fresh approach to the issue by integrating high-drive bus ports and control logic in one package. The chip contains a 600-gate equivalent programmable logic array and tridirectional (three-way) bus transfer logic, housed in a 44-pin plastic leaded chip carrier. The packaging and, behind the scene, a proprietary CMOS II-E EPROM process cut board space by three or four times compared with discrete circuits.

The CMOS process also enhances the basic attributes called for in bus interface logic: speed and high drive current. The maximum data delay between ports is 25 to 35 ns; and the array can operate in TTL or CMOS systems at clock speeds to 12.5 MHz. The high-drive ports can sink 16 mA from a 300-pF load. The EPROM technology also permits 100% device testing.

Another benefit of the chips that TTL devices do not offer is an optional "zero-standby" power mode. If the chip's inputs are static for more than 50 to 100 ns, the device powers down from its normal operating current of about 108 mA at 1 MHz to its quiescent current of several microamps.

The controller's two major functional blocks are the bus management unit and the programmable logic unit. The two circuits communicate by way of internal signal paths that replace the external logic and traces of conventional interfaces.

The device has five 8-bit ports (Fig. 1). Three ports on the bus-management unit make possible three-way asynchronous data flow, and thus are equivalent to three bidirectional chips. The two other ports are a dedicated input and an I/O port for the programmable logic unit.

DATA ROUTED THROUGH THREE PORTS

The bus-management unit's main task is to route real-time or transparent-latched data, which can be inverted. The unit does this through the three ports, which can be programmed as inputs or outputs. A direction-control unit dynamically selects the desired port and routes the data accordingly.

In addition, the bus unit can send latched data from any input port to the programmable logic unit, whose architecture is similar to that of EPROM-based EPLDs. The primary difference is the addition of up to eight buried registers, asynchronous controls, and an extra set of inputs from the three bus-management-unit ports. In all, inputs to the programmable logic unit include a dedicated port to the input logic macrocells, a bus unit feedback path, an I/O port, and feedback from I/O logic macrocells.

A conventional sum-of-products array and a logic macrocell structure perform the logic unit's control functions. The user programs the input and I/O logic macrocells to supply either latched or real-time data. Polarity is also determined by the

"Reprinted with permission from Electronic Design (Vol. 35, No. 3) February 5, 1987. Copyright 1987 Hayden Publishing Co., Inc., a subsidiary of VNU."

user, permitting the device to handle either active-high or active-low outputs.

Besides its flexible input structure, the logic array in every I/O macrocell offers 13 product terms, each equal to a 64-input AND gate. Sequential and combinatorial logic are possible, since the macrocells contain registers (Fig. 2). Data can also feed back to the array. The user configures the macrocell for a specific task by choosing among positive-edge-triggered D, T, SR, and JK flip-flops.

For maximum control, an asynchronous clock can be derived from one to eight product terms. Each flip-flop also has an asynchronous set and reset. A programmable Output Enable signal selectively enables outputs to emulate open-collector operation.

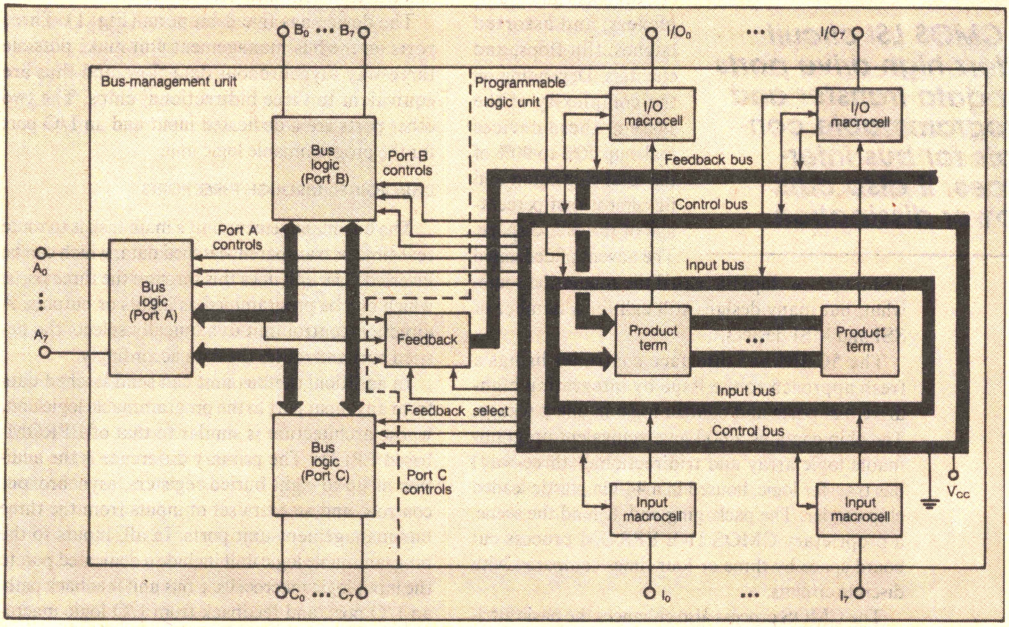
Three buses carry the chip's internal communication and control signals. The input bus serves the input macrocells, as well as the I/O macrocells configured as inputs. The feedback bus provides bus unit and I/O-macrocell feedback. Finally, the control bus steers the data through the various ports connecting the bus management and programmable logic units. It also supplies complex control signals for the bus functions, like Direction Control, Latching,, and Output Enable, as well as Clock, Set, Reset, Latching and Output Enable source signals for the logic unit.

Although the bus-interface controller has several advantages over conventional TTL interfaces, designers might question the time needed to develop and program complex LSI devices. That time, however, is cut to minutes through use of the Programmable Logic Development System software, known in its second revision as iPLDS-II. The system includes several modules that minimize design and development time. The modules offer high-level primitives that define the design and serve as logic building blocks.

To best apply the bus controller in a system, a designer exploits its high level of integration. A system with a dual-port memory shared by two processors offers a good example. Such designs are helpful when resources, such as large amounts of data in a memory, must be shared.

Data flowing between the memory and the two microprocessors needs buffering and arbitration to prevent contention. The usual solution is for data transceivers to isolate the data from the shared memory and for a controller to arbitrate the demands of the processors.

However, the bus-interface controller performs these functions with the minimum chip count. The programmable logic unit arbitrates data requests according to a predefined protocol that establishes priorities, and the bus management unit's three ports isolate the data. Moreover,



1. With the 5CBIC bus interface controller, a bus management unit communicates with a programmable logic unit through a segmented bus structure. The feedback and control buses link the two units and the input bus feeds the array from the input and I/O macrocells.

the control section supplies feedback paths, eliminating chips that would be needed in a conventional design.

In the example, two cascaded controllers handle 16-bit-wide data. Since the system needs only one logic array, the second is available for whatever the designer may need, including an up/down octal counter; memory, I/O, or interrupt controller; or an addressable 8-bit register.

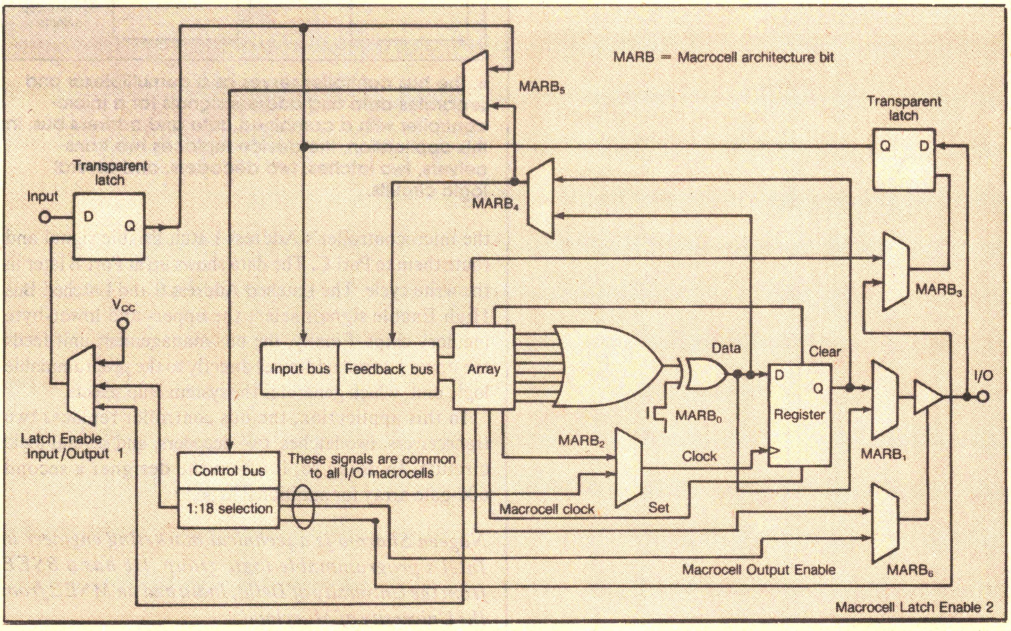
Ports A and B of the bus-management unit connect to the two processors, and Port C connects to the system memory (Fig. 3). If the processors run on different clocks, the controller must synchronize the access requests of one unit with those of the second. The programmable logic unit performs all sampling, synchronization, and arbitration, and the registers within it supply the control signals needed by the bus unit. The I/O pins serve as inputs without sacrificing the sum-of-products logic that is important to effective gate use.

The system's arbitration scheme is straightforward. Only one processor at a time may have access to the system bus, with processor 1 having the higher priority in the event of a simultaneous request. If the second processor requests an access while the first has control of the bus, wait states extend the second unit's bus cycle. When the first data transfer is completed, the programmable logic unit grants access to the second processor and enables the address latches and data transceivers.

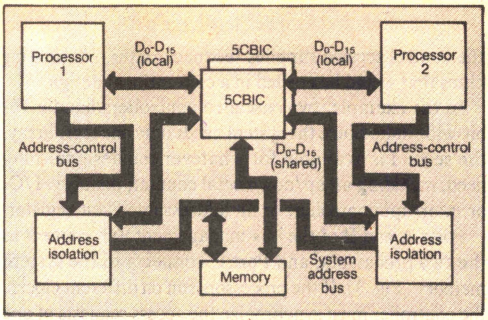
In a second example, the device functions as a demultiplexer for microprocessors, microcontrollers, and peripheral controllers with combined address and data buses. The time-multiplexed buses use package pins more efficiently, but the address and data signals must be separated before interfacing with memories or I/O devices.

Once again, two bus controllers accommodate 16-bit data, this time from an 8096 microcontroller (Fig. 4). In a system containing only a RAM, an EPROM, and I/O devices, the two A ports of the bus management units connect to the 8096's Ports 3 and 4. Ports B and C connect directly to the system data and address buses. In this application, Ports A and B are bidirectional, and Port C is an output only. As a result, Port A is the only input to Ports B and C, and Port B is the only input to Port A (during a memory-read cycle).

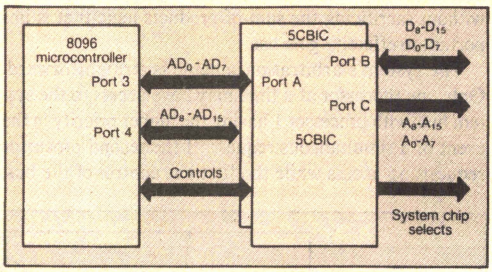
The user programs the chip to latch the addresses with



2. The input and I/O logic macrocells include architecture bits $MARB_0$ through $MARB_6$ and a choice of control signals that permit numerous configurations, such as latching, inversion, asynchronous clocking, set, reset, and output enable.



3. Two controller chips handle 16-bit-wide data in a system with a dual-port memory. The chips isolate data as well as supply signals for arbitration, synchronization, and external control.



4. The bus controller serves as a demultiplexer and separates data and address signals for a microcontroller with a combined data and address bus. In this application, the device replaces two transceivers, two latches, two decoders, and several logic circuits.

the microcontroller's Address Latch Enable signal and route them to Port C. The data shows up at Port B later in the write cycle. The Latched Address 0 and Latched Bus High Enable signals select the upper- and lower-byte memory chips. Finally, the bus-management unit feeds the upper latched addresses directly to the programmable logic unit, which generates the system chip selects.

In this application, the bus controller replaces two transceivers, two latches, two decoders, and several logic circuits. In doing so, it offers the designer a second 600-gate array for additional logic. □

Nageen Sharma is a technical marketing engineer in Intel's programmable logic group. He has a BSEE from the University of Delhi, India and an MSEE from the University of Maryland.

Development Support Tools 4

PRELIMINARY

iPLDS II
THE INTEL PROGRAMMABLE LOGIC DEVELOPMENT SYSTEM VERSION II

- **Hardware and Software Necessary to Turn Design Concepts into Functional Erasable Programmable Logic Devices (EPLDs)**
- **Menu-driven Software with On-line Help Messages for All Stages of the Design Process**
- **iUP-PC Hardware Programs Intel EPLD's, EPROM's, E²PROM's, Peripherals, and Microcontrollers with one PC-based System**
- **All Equipment Interfaces with the IBM PC/XT*, PC/AT*, and True Compatibles**
- **JEDEC Standard Design File, Part Utilization Report, Minimized Equation File, and Compiler Error File All Available as Outputs**
- **Supports a Variety of Input Methods:**
  - **Schematic Entry**
    - **— TTL Library**
    - **— EPLD Primitives Library**
  - **Text Editor Entry**
    - **— State Machine**
    - **— Boolean Equations**
- **Macro Expander Accepts TTL, and User-Defined Macros and Expands Them into Equivalent EPLD Primitives**
- **Espresso** Minimizer Reduces Logic Equations to Least Number of Product Terms**
- **Supports All Intel EPLD's Including the 5CBIC and 5AC312**

Release 1.5 of Intel's Programmable Logic Development System II (iPLDS II) is a powerful set of tools for transforming a logic design into customized silicon. The system provides design entry, logic compilation, and device programming capability on a desktop using an IBM PC/XT, PC/AT, or compatible.

290134-1

iPLDS II Components Picture

*IBM PC/XT, PC/AT are registered trademarks of International Business Machines Corporation.

**ESPRESSO is a copyrighted by the University of California at Berkeley and is used with permission.

November 1987
Order Number: 290134-003

INTRODUCTION TO PROGRAMMABLE LOGIC DESIGN

When performing a programmable logic design on a CAD system, the design must first be entered using one of a variety of entry methods. These methods typically include schematic capture or Boolean equation entry using a standard text editor. Less typical entry methods include netlist entry, whereby a hand drawn schematic can be entered in a node-by-node fashion, or state machine entry in a text or graphical mode.

Once the design has been entered into the CAD package, several processing steps may occur. The design is usually translated into a format usable by the software, logic reduction may be performed, and, finally, some form of programming file can be produced. Most CAD packages also produce documentation of the minimization and device fitting results, including the final pin assignments.

Once the programming file has been generated, the design can be transferred into silicon in a programming manner similar to that used for EPROMs.

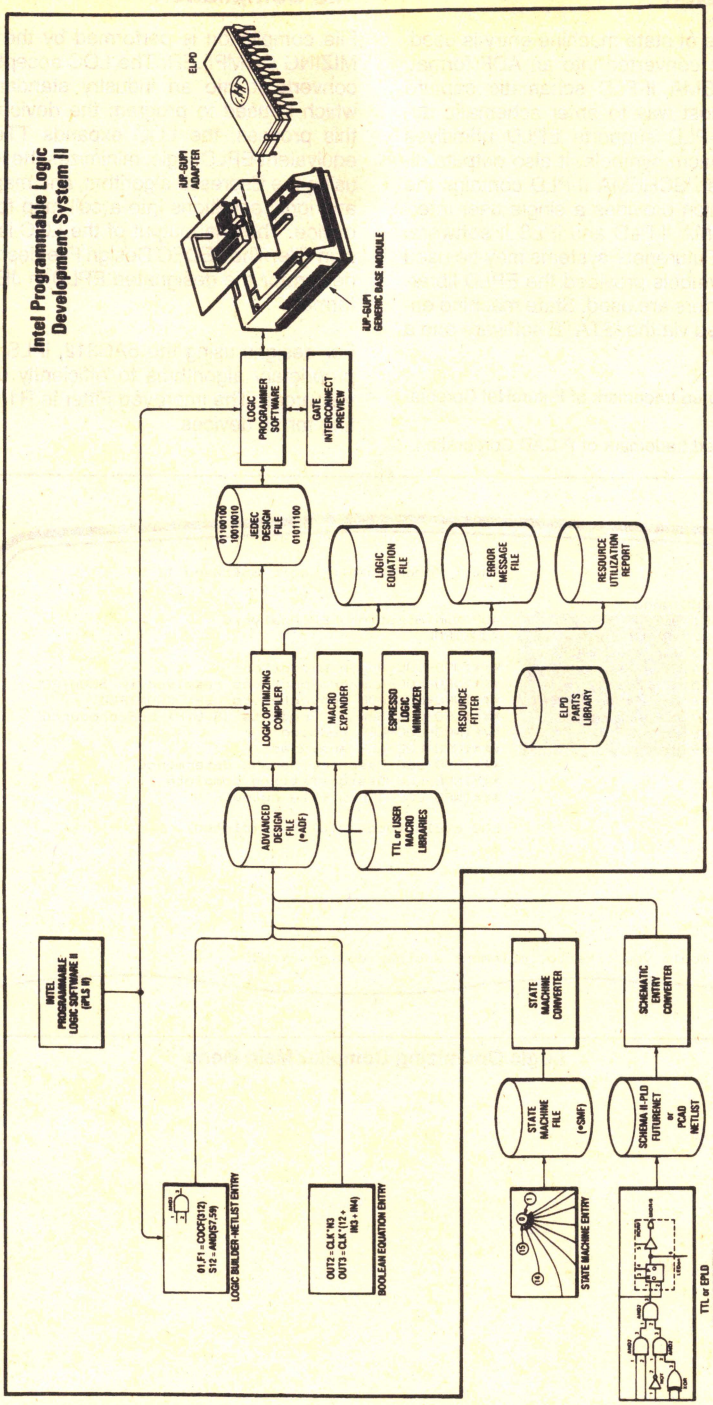
FUNCTIONAL DESCRIPTION OF iPLDS II

All of the design entry methods with the exception of graphic state machine entry are supported by the iPLDS II software. iPLDS II supports netlist and Boolean equation entry using any standard text editor. State machine software and schematic capture libraries are also available from Intel as optional entry methods. Depending on the entry format used, the design may require translation into Advanced Design File (ADF) format. Once the design is in ADF form, the Logic Optimizing Compiler expands any TTL macros, minimizes all equations, and fits the design into a device-specific JEDEC Design File. The JEDEC Design File is programmed into the EPLD by the Logic Programmer Software using the iUP-PC hardware. Thus, the circuit design is transformed into an operating EPLD on one workstation.

The Intel Programmable Logic Software II (iPLS II) is composed of four functional modules: design entry, netlist conversion, file compilation and device programming.

Design Entry

Design entry is typically accomplished by creating an ADF using an ASCII text editor, or by using a schematic capture package.


Intel Programmable Logic Development System II
290134-2
EPLD
iUP-SUPI ADAPTER
iUP-SUPI GENERIC BASE MODULE
LOGIC PROGRAMMER SOFTWARE
GATE INTERCONNECT PREVIEW
JEDEC DESIGN FILE
LOGIC EQUATION FILE
ERROR MESSAGE FILE
RESOURCE UTILIZATION REPORT
LOGIC OPTIMIZING COMPILER
MACRO EXPANDER
ESPRESSO LOGIC MINIMIZER
RESOURCE FITTER
EPLD PARTS LIBRARY
ADVANCED DESIGN FILE (*.ADF)
TTL or USER MACRO LIBRARIES
INTEL PROGRAMMABLE LOGIC SOFTWARE II (PLS II)
STATE MACHINE CONVERTER
SCHEMATIC ENTRY CONVERTER
STATE MACHINE FILE (*.SMF)
SCHEMA II-PLD FUTURENET or PCAD NETLIST
LOGIC BUILDER-NETLIST ENTRY
BOOLEAN EQUATION ENTRY
STATE MACHINE ENTRY
TTL or EPLD SCHEMATIC ENTRY

Netlist Conversion

If schematic capture of state machine entry is used, the design must be converted into an ADF format. The optional SCHEMA II-PLD schematic capture package is a low-cost way to enter schematic designs. SCHEMA II-PLD supports EPLD primitives and user-defined macro symbols. It also outputs directly in ADF format. SCHEMA II-PLD contains the EPLD Manager, which provides a single user interface to both SCHEMA II-PLD and iPLS II software. The P-CAD†† and Futurenet† systems may be used to capture EPLD symbols provided the EPLD libraries and ADF convertors are used. State machine entry may be performed via the iSTATE software and a standard text editor.

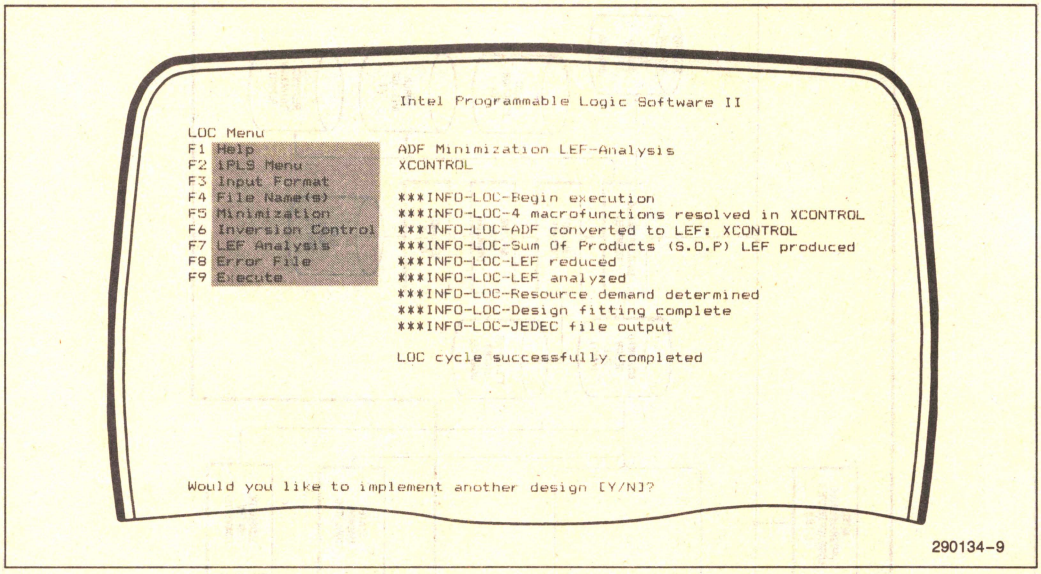
†Futurenet is a registered trademark of FutureNet Corporation.

††P-CAD is a registered trademark of P-CAD Corporation.

File Compilation

File compilation is performed by the LOGIC OPTIMIZING COMPILER. The LOC accepts an ADF and converts it into an industry standard JEDEC file which is used to program the device. As a part of this process, the LOC expands TTL macros into equivalent EPLD logic, minimizes the logic equations using the Espresso algorithm, and maps the network and logic equations into a cell map for the selected device. The final output of the LOC is a JEDEC Design File. The JEDEC Design File describes the input design for the designated EPLD in JEDEC standard format.

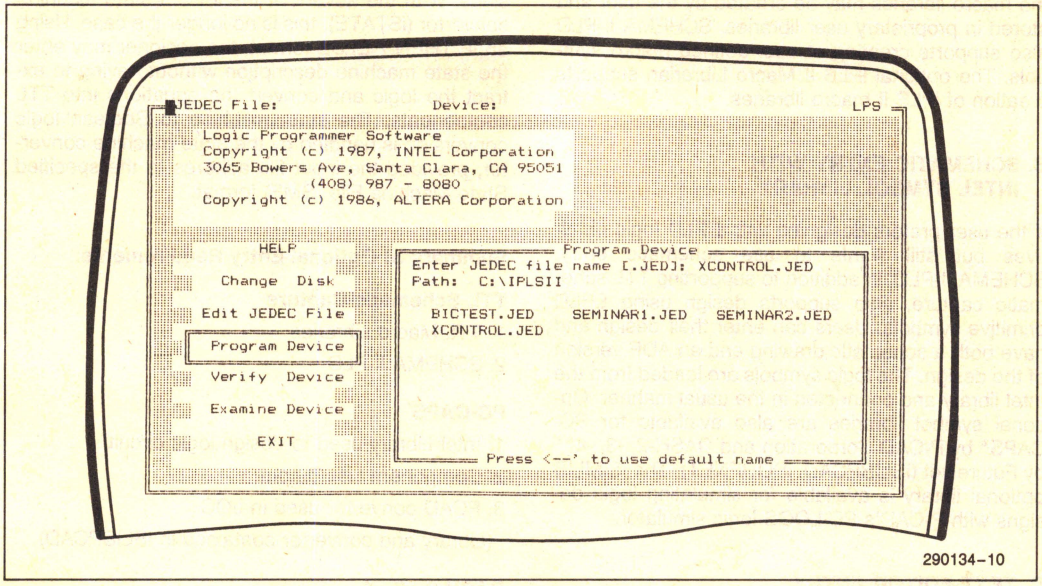
For designs using the 5AC312, iPLS II R1.5 utilizes proprietary algorithms to efficiently use the device resources. The improved Fitter in R1.5 optimizes fitting for all devices.



Logic Optimizing Compiler Main Menu

Device Programming

The programming hardware is controlled by the LOGIC PROGRAMMER SOFTWARE. LPS takes the JEDEC file produced by the LOC and programs it into the device. LPS can also read a programmed device or verify that a device has been programmed correctly.



Logic Programmer Software Main Menu

The Intel Universal Programmer for the Personal Computer (iUP-PC) is a versatile programming solution in a PC-based system. Installed in an IBM PC/XT, PC/AT or compatible host, the iUP-PC emulates the performance of the standalone INTEL iUP-200A Universal Programmers. As such, it supports the iUP Generic Universal Programmer Interface (iUP-GUPI). With the appropriate socket adapters for the iUP-GUPI, the iUP-PC supports all Intel EPLDs. Future EPLDs will be supported by new GUPI adapters or adapter upgrades. Other Intel devices—EPROMs, EEPROMs, and microcontrollers—are also supported by the GUPI. The iUP-PC is controlled by the LPS or the iPPS (Intel PROM Programmer Software). iPLDS II includes the iUP-PC, which contains the iPPS, PCPP programming card, interconnect cable, and the GUPI base. GUPI adapters are available separately.

iPLS II SOFTWARE

The Intel Programmable Logic Software II (iPLS II) has many options and enhancements for implementing a logic design. iPLS II accommodates a wide variety of design input methods. Schematics, state machines or Boolean equations may all be used provided the proper formats and convertors are implemented as needed. No matter what method is chosen, the Logic Optimizing Compiler will minimize and fit the design during compilation. Finally, iPLS II contains the Logic Programmer Software which controls the iUP-PC programming hardware for all Intel EPLDs.

I. Design Input

The entire spectrum of design input methods is available to the logic designer in iPLS II. Everything from TTL schematics to Boolean equations are accepted and processed by the LOC.

A. TTL SCHEMATIC ENTRY

SCHEMA II-PLD is an optional software package that allows EPLD design to be implemented with standard TTL functions. SCHEMA II-PLD contains a symbol library that includes common SSI/MSI TTL symbols. SCHEMA II-PLD also outputs directly in ADF format. The TTL symbols appear in the ADF in the form of macro calls. During compilation, iPLS II automatically expands these calls from its TTL macro library. Thus, with SCHEMA II-PLD, conversion to EPLD logic primitives is performed automatically in a manner completely transparent to the user.

Only parts supported by the SCHEMA II-PLD TTL symbol library and the iPLS II TTL macro definition library may be used for TTL schematic entry. In most cases, this won't be a limitation as the most common parts are included in both libraries. Parts not in the macro libraries may be created by the user and stored in proprietary user libraries. SCHEMA II-PLD also supports creating of user-defined macro symbols. The optional iPLS II Macro Librarian supports creation of iPLS II macro libraries.

B. SCHEMATIC ENTRY WITH INTEL SYMBOL LIBRARY

If the user prefers designing with EPLD logic primitives but still wants to use schematic entry, SCHEMA II-PLD, in addition to supporting TTL schematic capture, also supports design using EPLD primitive symbols. Users can enter their design and have both a schematic drawing and an ADF version of the design. The logic symbols are loaded from the Intel library and connected in the usual manner. Optional symbol libraries are also available for PC-CAPS* by P-CAD Corporation and DASH-2, -3, -4** by FutureNet (iSLIBPCAD, iSLIBFNET). The iSIMLIB optional library is available for simulating logic designs with P-CAD's PC-LOGS logic simulator.

C. TEXT EDITOR ENTRY

Designers who are familiar with the logic primitives and the Advanced Design File format can directly enter ADFs with a standard text editor. The bulk of the design entry can be accomplished using Boolean Equations obtained from a Karnaugh map or truth table. Hence, the need for conversion to gates is eliminated. This method of entry is useful for sub-circuits that will be incorporated into larger designs.

*PC-CAPS and PC-LOGS are registered trademarks of P-CAD Corporation.

**DASH-2, -3, -4 are registered trademarks of FutureNet Corporation.

D. STATE MACHINE ENTRY

In the past, state diagrams or flowcharts (ASM charts) were merely abstractions used to obtain the logic equations necessary to implement TTL designs. With the advent of the iPLS II state machine convertor (iSTATE), this is no longer the case. Using an IF THEN / ELSE format, the designer may enter the state machine description without having to extract the logic and convert the equations into TTL components. The state machine to Boolean logic conversion is handled by the state machine convertor, provided the input file adheres to the specified State Machine File (SMF) format.

Summary of Optional Entry Requirements:

TTL Schematic Capture

1. TTL Macro Libraries
2. SCHEMA II-PLD

PC-CAPS

1. Intel Library used to design logic circuit
2. Component List Output
3. PCAD convertor used in LOC

   (Library and convertor contained in iSLIBPCAD)

DASH-2, -3, -4

1. Intel Library used to design logic circuit
2. Pin List Output
3. FutureNet convertor used in LOC

   (Library and convertor contained in iSLIBFNET)

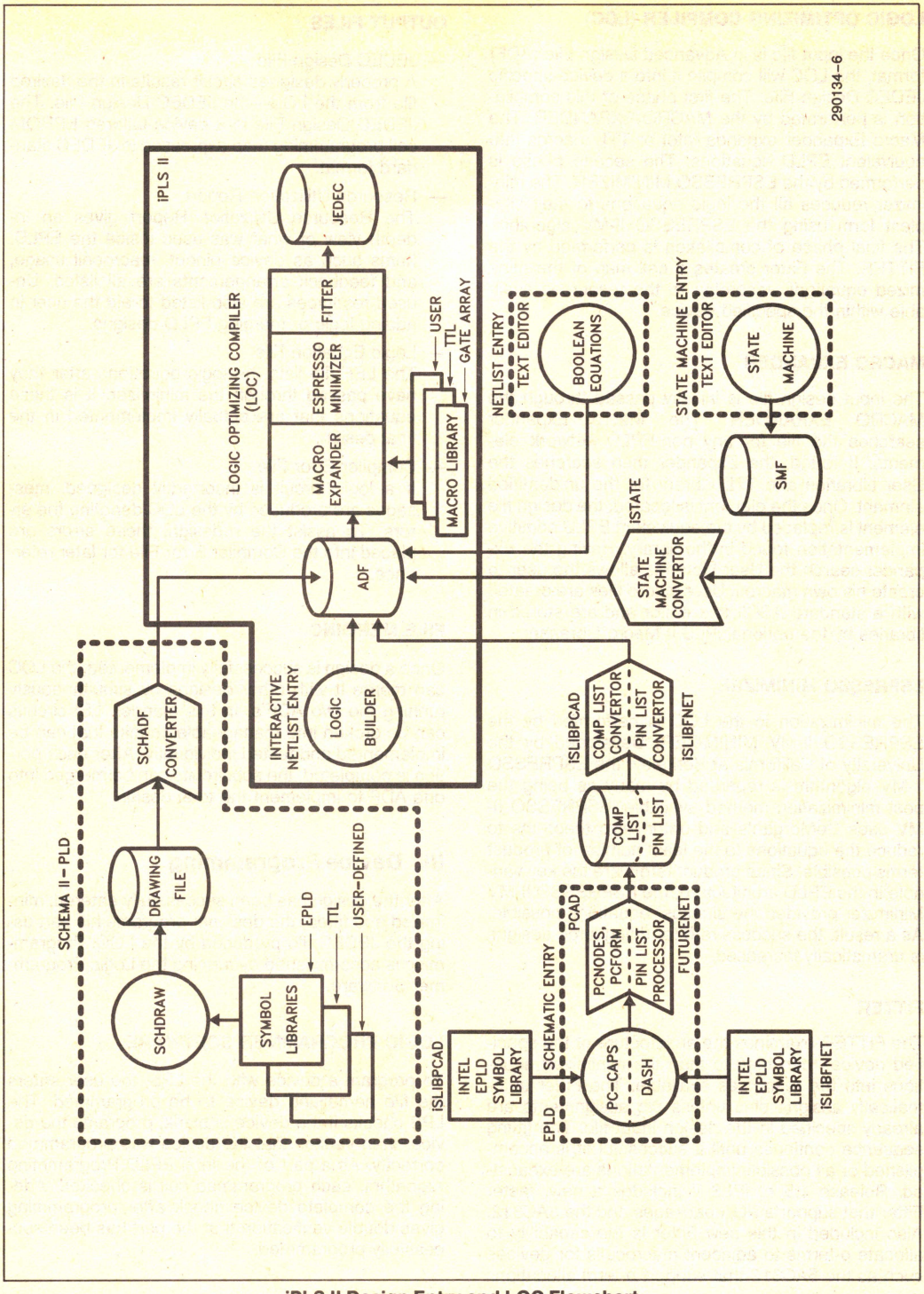
State Machines

1. State Machine File (SMF) format used
2. Optional state machine convertor used in LOC

   (Convertor contained in iSTATE)

II. Logic File Compilation

Before programming the part, the designer must compile the input design file into a JEDEC standard file. This function is performed by the Logic Optimizing Compiler.



iPLS II Design Entry and LOC Flowchart

LOGIC OPTIMIZING COMPILER (LOC)

Once the input file is in Advanced Design File (ADF) format, the LOC will compile it into a device-specific JEDEC Design File. The first phase of this compilation is performed by the MACRO EXPANDER. The Macro Expander expands Intel or TTL macros into equivalent EPLD equations. The second phase is performed by the ESPRESSO MINIMIZER. The minimizer reduces all the logic equations to their simplest form using the ESPRESSO II-MV algorithm. The final phase of compilation is performed by the FITTER. The Fitter creates a cell map of the minimized equations according to the resources available within the specified device.

MACRO EXPANDER

The input design file is initially passed through the MACRO EXPANDER. The Macro Expander searches the file for any non-EPLD network elements. If found, the Expander then searches the User Libraries and TTL Library for the unidentified element. Once the element is located, the design file element is replaced by the equivalent EPLD primitive implementation found in the library. Having the Expander search the User Libraries allows the user to create his own macros. User macro files are created with a standard ASCII text editor and are stored in libraries by the optional iPLS II Macro Librarian.

ESPRESSO MINIMIZER

The minimization in the LOC is performed by the ESPRESSO II-MV MINIMIZER. Developed by the University of California at Berkeley, the ESPRESSO II-MV algorithm is regarded by many as being the best minimization method available. ESPRESSO II-MV uses DeMorgan's and other logic theorems to reduce the equations to the least number of product terms possible. Since product terms are the key variable in the EPLD architecture, the ESPRESSO II-MV Minimizer provides the simplest equations possible. As a result, the success rate for fitting large designs is dramatically increased.

FITTER

The FITTER examines the architecture of the specified device, then tries to map the minimized equations into the resources available. The Fitter automatically assigns pins unless pin assignments are already specified in the design input file. The fitting sequence continues until a successful fit is accomplished or all possible implementations are exhausted. Release 1.5 of iPLS II includes a new, faster Fitter that supports PGA packages and the 5AC312. Also included in this new Fitter is the capability to allocate p-terms to adjacent macrocells for devices such as the 5AC312 that support p-term allocation.

OUTPUT FILES

— JEDEC Design File
A properly designed circuit results in the desired file from the LOC—the JEDEC Design File. The JEDEC Design File is a device-tailored EPROM cell programming map expressed in JEDEC standard format.

— Resource Utilization Report
The Resource Utilization Report gives an in-depth view of what was used inside the EPLD. Items such as device pinout, macrocell usage, and feedback arrangements are all listed. Unused resources are also listed to aid the user in adding logic or merging EPLD designs.

— Logic Equation File
The LEF file lists the logic equations after they have passed through the minimizer. It is these equations that are actually implemented in the final design.

— Compiler Error File
If a logic circuit is incorrectly designed, messages are produced by the LOC denoting the errors. To assist the redesign, these errors are placed into the Compiler Error File for later reference.

FILE MERGING

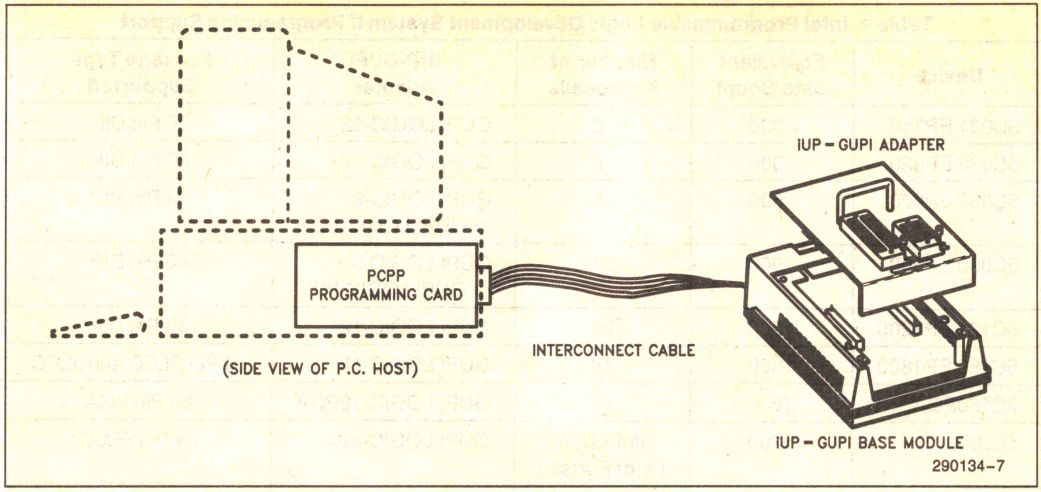
Once a design is successfully implemented, the LOC can merge it with other designs by simultaneously running the two ADF's. In this manner, LSI circuits can be broken into manageable chunks that can be implemented and tested individually. After each portion is completed, the subcircuits can be merged into one ADF to implement the total design.

III. Device Programming

After the design has been successfully entered, minimized and fitted, the designer programs his part using the JEDEC file produced by the LOC. Programming is accomplished by running the Logic Programmer Software.

LOGIC PROGRAMMER SOFTWARE

To program a device with the LPS, the user enters the file name and device to be programmed. The LPS checks if the device is blank, programs the device, then verifies that the device was programmed correctly. As a part of the Intel EPLD Programming Algorithm, each programmed cell is checked. Adding the complete device check after programming gives double verification that the part has been successfully programmed.



The Intel Universal Programmer for the Personal Computer (iUP-PC)

It is also possible to read a pre-programmed device and program other devices with the program read. The JEDEC Editor in LPS provides a hierarchical view of the device from the pin level, to the macrocell level, to the product term level. At the product term level, individual EPROM cells may be set or reset to connect or disconnect the logic equation inputs.

If the user does not want an EPLD to be read, the Security bit may be set when running the LPS. The Security Bit prevents a device from being examined after it has been programmed. This function is useful for protecting confidential designs.

iUP-PC HARDWARE

The Intel Universal Programmer for the Personal Computer consists of the PCPP programming card, 50-lead interconnect cable, GUPI base and GUPI adapter. Together they form a system for programming most PROM-type Intel devices directly from the PC host.

PCPP

The Personal Computer Personal Programmer (PCPP) is the programmer interface card that fits into the IBM AT/XT or true compatible. It is capable of driving both the iUP-GUPI base and the iUP-FAST27K personality module. The PCPP emulates the performance of the Intel iUP-200A. The LPS or iPPS (Intel PROM Programmer Software) controls the PCPP, causing the programming card to generate the control signals for the GUPI base.

GUPI BASE

The Generic Universal Programmer Interface (GUPI) is used for all programmable logic support. As all signal generation to devices is done by the GUPI, the programming waveforms are extremely reliable. Using the GUPI also allows upgrading for future devices with the simple addition of a plug-in adaptor. Future Intel EPLDs will be supported by the GUPI system.

GUPI ADAPTERS

Table 1 details the GUPI adapters required for the logic devices. The adapters available for programming EPROM's, E^2PROM's and microcontrollers can be found in the data sheet for the iUP-PC (Intel order number 290130). The adapters contain the device description data for a family of similar devices. New devices will be supported by new adapters or by a firmware upgrades in existing adapters.

SPECIFICATIONS

Host System

The iPLDS II software requires an IBM PC/XT, PC/AT or other true compatible computer capable of running MS-DOS* version 2.0 or later. The computer must have a 360KB double-sided, double-density diskdrive, a hard disk, and 512KB of RAM. Additional memory is required for the optional schematic capture programs. A color monitor is recommended,

Table 1. Intel Programmable Logic Development System II Programming Support

Device	Equivalent Gate Count	Number of Macrocells	iUP-GUPI Adapter	Package Type Supported
5C031 EP310	300	8	GUPI LOGIC-12	20 Pin DIP
5C032 EP320	300	8	GUPI LOGIC-12	20 Pin DIP
5C060 EP600	600	16	GUPI LOGIC-9 or GUPI LOGIC-IID	24 Pin DIP
5C090 EP900	900	24	GUPI LOGIC-9 or GUPI LOGIC-IID	40 Pin DIP
5C121 EP1200	1200	28	GUPI LOGIC-12	40 Pin DIP
5C180 EP1800	1800	48	GUPI LOGIC-18	68 Pin PLCC and JLCC
5C180PGA	1800	48	GUPI LOGIC-18PGA	68 Pin PGA
5CBIC	1200	(inPLU):8 (# of Ports):5	GUPI LOGIC-BIC	44 Pin PLCC
5AC312	1200	12	GUPI LOGIC-IID	24 Pin DIP

(EPXXX Devices from Altera Corp.)

as the color graphics available provide a better representation of the data than a monochrome display. The PCPP programming card requires one full-size card slot in the host computer.

*MS-DOS is a trademark of Microsoft Corporation

Operating Environment

Electrical Characteristics

PCPP: Worst Case Power Consumption at IBM PC I/O Channel

Supply Voltage	Voltage Variance	Personality Module	Max. Current Drain
+5V	+5%, −4%	FAST27K	1.898 A
−12V	+10%, −9%	FAST27K	102.9 mA
+12V	+5%, −4%	GUPI	530 mA

Physical Characteristics

PCPP:

Length: 13.3 inches (33.9 cm)
Height: 3.9 inches (10.0 cm)

INTERCONNECT CABLE:

50 lead ribbon cable
Length: 3.0 feet (91.4 cm)
Width: 2.43 inches (5.5 cm)

GUPI:

Length: 7.0 inches (17.8 cm)
Width: 5.5 inches (1.4 cm)
Height: 1.6 inches (4.1 cm)

Environmental Characteristics

Operating Temperature: 10°C to 40°C
Operating Relative Humidity: 85% Maximum

Equipment Supplied

HARDWARE

— PCPP programming card
— Interconnect cable
— GUPI base
— (GUPI-LOGIC adaptors purchased separately)

SOFTWARE

— iPLS II (5 diskettes)
— iPPS (2 diskettes)

DOCUMENTATION

— iPLS II User's Guide (order number 450196)
— iPLS II Release 1.5 Supplement (order number #453703)
— PCPP User's Guide (order number 168161)

ORDERING INFORMATION

Order Code	Product Description
iPLDS II	Intel Programmable Logic Development System II: iPLS software, iUP-PC, iPLS II User's Guide
iPLS II	Intel Programmable Logic Software II: Logic Builder design entry, Logic Optimizing Compiler, Logic Programmer Software, iPLS II User's Guide
iUP-PC	Intel Universal Programmer for the Personal Computer: PCPP programming card, interconnect cable, iUP-GUPI base, Intel PROM Programming Software PCPP User's Guide
MLIB	iPLS II Macro Librarian: Macro Librarian Software and User's Guide Supplement for creating user-defined macro libraries.
iSTATE	Intel State Machine Software: Entry format documentation, state machine convertor for LOC
iSLIBFNET	Intel Symbol Library—FutureNet: EPLD symbol library for FutureNet DASH-2 schematic capture package, Futurenet Pinlist convertor for LOC
iSLIBPCAD	Intel Symbol Library—PCAD: EPLD symbol library for PCAD PC-CAPS schematic capture package, PCAD Component List convertor for LOC
iSIMLIB	Intel Simulation Library (PC-LOGS): EPLD simulation library for PC-LOGS simulator by PCAD
iUP-GUPI	Intel Universal Programmer—Generic Universal Programmer Interface: Generic programmer base which holds GUPI adaptors
GUPI LOGIC-IID	GUPI Adaptor for the 5AC312, 5C060, 5C090, and future 24-pin and 40-pin EPLDs.
GUPI LOGIC-09	GUPI Adaptor for the 5C060 and 5C090 DIP EPLDs
GUPI LOGIC-12	GUPI Adaptor for the 5C031, 5C032, 5C121 and future 20 pin DIP EPLDs
GUPI-LOGIC-18	GUPI Adaptor for the 5C180 and future 68 pin PLCC and JLCC EPLDs
GUPI LOGIC-18PGA	GUPI Adaptor for the 5C180 device in a 68 pin PGA package.
GUPI-LOGIC-BIC	GUPI Adaptor for the 5CBIC and follow-on products
ADAPT24TO28	Adapts 24 pin DIP socket to 28 pin PLCC socket; for use with GUPI LOGIC-09 and GUPI LOGIC-IID.
ADAPT40TO48	Adapts 40 pin DIP socket to 44 pin PLCC socket; for use with GUPI LOGIC-09 and GUPI LOGIC-IID.

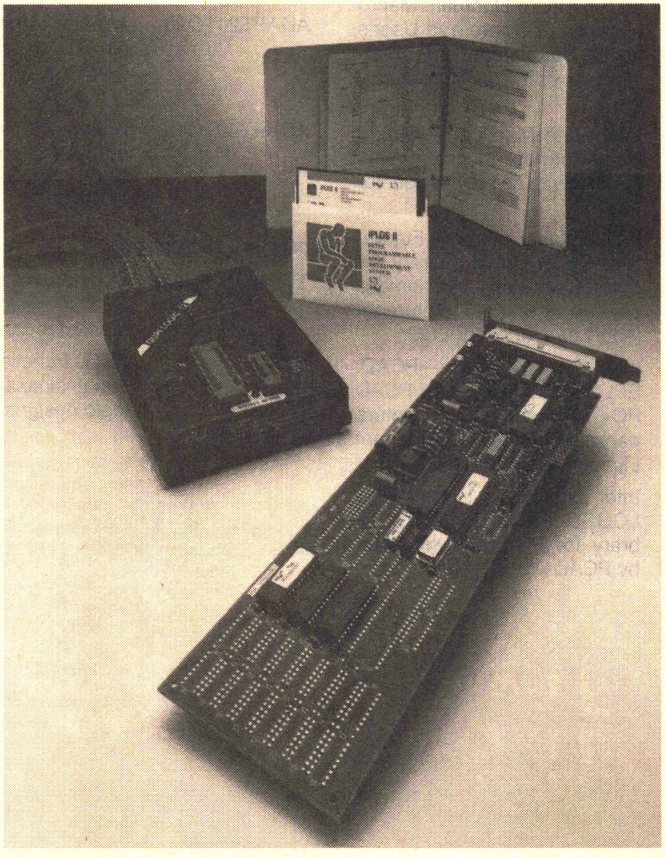
iPLDS II UTILITIES

TTL Macro Library	TTL Macro Library Macros accessed by Macro Expander for most standard TTL symbols
SIM	EPLD Functional Simulator Functional simulator for most EPLD logic designs.

iUP-PC
INTEL UNIVERSAL PROGRAMMER FOR THE PERSONAL COMPUTER

- **Personal Computer Version of the iUP-200A/201A Universal Programmers**
- **Runs on an IBM PC/AT*, PC/XT* or True Compatible**
- **GUPI and FAST27K Personality Modules Provide Support for Numerous Device Families**
- **Utilizes the int$_e$ligent™ and Quick-Pulse Programming™ Algorithms**
- **Easily Upgradable for new Devices Through Low-Cost Plug-In Adapters**
- **Extremely Versatile—Programs Intel or Intel-Compatible EPROM, E^{2}PROMs, EPLDs, Peripherals and Micro-Controllers, Including the New 5AC312 EPLD**

The Intel Universal Programmer for the Personal Computer, iUP-PC, provides a high performance programming solution from a PC host. Through plug-in adapters for the Generic Universal Programmer Interface (iUP-GUPI), the iUP-PC supports virtually all programmable Intel devices. Upgrades for new devices are made by the simple addition of a GUPI adapter or the upgrade of an existing adapter.



290130-1

NOTE:
GUPI Adapter NOT included.
*IBM PC/AT and PC/XT are registered trademarks of International Business Machines Corporation.

November 1987
Order Number: 290130-002

FUNCTIONAL DESCRIPTION

The iUP-PC provides a fast, versatile and reliable programming solution from a Personal Computer host. Downloading to a stand-alone programmer or moving from one workstation to another is no longer required. With the iUP-PC, the designer may do his development and programming on one workstation. Through the Generic Universal Programmer Interface (iUP-GUPI), the iUP-PC is made extremely versatile. With the iUP-GUPI the designer may program across EPROM, E^2PROM, Microcontroller, Peripheral and EPLD device categories with the mere change of a plug in adapter. No other hardware or software addition is needed. As all of the programming signals are generated at the GUPI base, extremely reliable waveforms reach the device.

COMPONENTS

The iUP-PC programming system consists of five components:

PCPP—The Personal Computer Personal Programmer (PCPP) is an IBM PC/XT form factor expansion card which fits into an IBM PC/XT, PC/AT or true compatible.

Interconnect Cable—A 50 lead ribbon cable connects the PCPP to the iUP-GUPI.

iUP-GUPI—The Intel Universal Programmer—Generic Universal Programmer Interface (iUP-GUPI) is the programming base which holds the device adapters.

GUPI Adapters*—The GUPI Adapters plug-in to the iUP-GUPI base. They carry the sockets and hardware for a particular device family.

iPPS—The Intel PROM Programmer Software (iPPS) runs on a personal computer under DOS and controls the PCPP/host communication.

***NOTE:**
Though the iUP-GUPI base is included in the iUP-PC package, the GUPI Adapters are NOT included. The desired adapters must be ordered separately.

PCPP CARD

The PCPP is an 8085 based co-processor board. Communication between the host and the PCPP may be controlled by the iPPS or LPS (Logic Programmer Software). Version 2.3 or greater of iPPS is required for running the iUP-PC on a personal computer. LPS is the programming software included in Intel's Programmable Logic Software II (iPLS II).

The PCPP is capable of driving the iUP-GUPI and FAST27/K modules. Future Intel devices will be supported by an iUP-GUPI adapter or adapter upgrade.

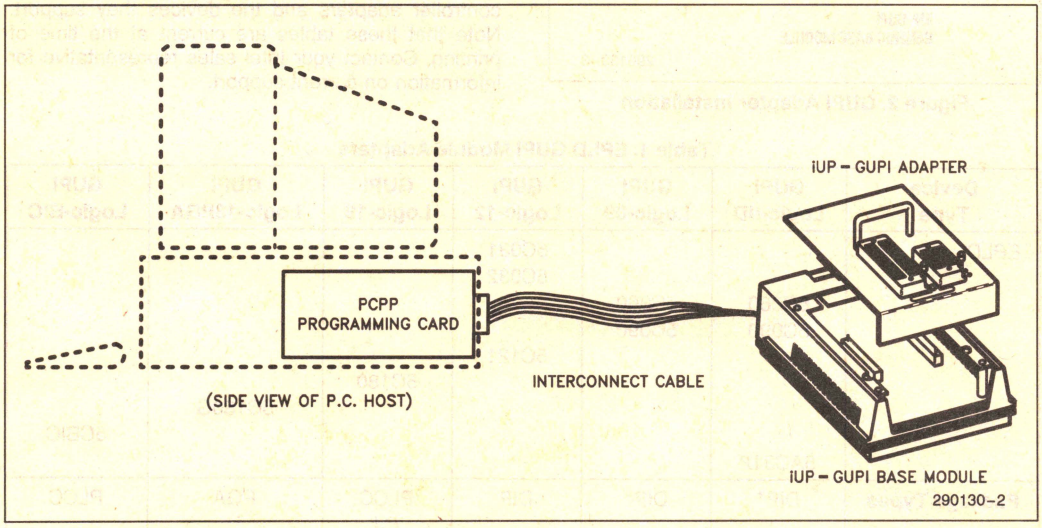


Figure 1. The Intel Universal Programmer for the Personal Computer (iUP-PC)

iUP-GUPI MODULE

The iUP-GUPI is a generic base module that enables the iUP-PC system to accept low-cost plug-in adapters. These adapters configure the system to support a wide variety of programmable devices—EPROMs, microcontrollers, and EPLDs—as well as device package types (refer to Table 1).

The iUP-GUPI module connects to the PCPP card via a ribbon cable. An opening in the top of the iUP-GUPI provides easy plug-in installation of the GUPI adapters (refer to Figure 2).

The iUP-GUPI offers the programming performance of earlier Intel personality modules, with the fastest Intel programming algorithms for each device type. For example, the iUP-GUPI uses the new Quick-Pulse Programming algorithm to program the 1-Meg EPROM in seconds.

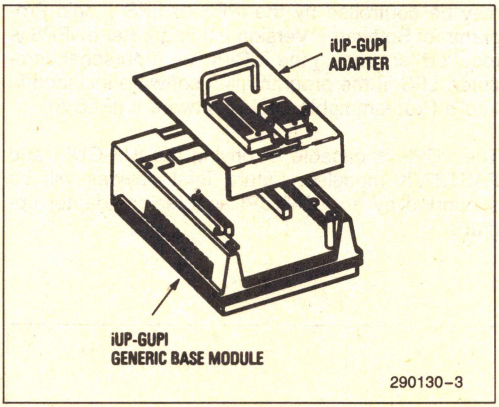


Figure 2. GUPI Adapter Installation

GUPI ADAPTERS

The iUP-GUPI adapters provide the final link of the iUP-PC programming system. The adapters provide the proper sockets and characteristic information for families of Intel devices.

The iUP-GUPI LOGIC adapters complete the programming solution of the Intel Programmable Logic Development System II (iPLDS II). The GUPI LOGIC adapters provide support for the entire range of Erasable Programmable Logic Devices (EPLDs). The adapters support families EPLDs with similar architecture, such as the 5C060 and 5C090. All future EPLDs will be supported by the GUPI LOGIC adapter system.

Intel's one megabit EPROMs are also supported with GUPI adapters. Adapters are available for the 27010, 27011, and 27210. The page mode of the 27011 is supported by the GUPI 27011 adapter. Other Intel EPROM support is provided with the FAST27/K personality module.

The MCS-51 and MCS-96 microcontroller families are supported by the GUPI MSC-51 and GUPI 8796 adapters. Supplemental adapters provide support for the variety of microcontroller package types. The 8741 and 8742 peripheral components are supported by the GUPI 8742 adapter.

Table 1 displays a cross-reference of the EPLD GUPI adapters and the devices they support. Table 2 displays a cross-reference of the EPROM/Microcontroller adapters and the devices they support. Note that these tables are current at the time of printing. Contact your Intel sales representative for information on current support.

Table 1. EPLD GUPI Module Adapters

Device Type	GUPI Logic-IID	GUPI Logic-09	GUPI Logic-12	GUPI Logic-18	GUPI Logic-18PGA	GUPI Logic-BIC
EPLD	5C060 5C090 5AC312	5C060 5C090	5C031 5C032 5C121	5C180	5C180G	5CBIC
Package Types	DIP*	DIP*	DIP	PLCC CJ	PGA	PLCC

*ADAPT units available to adapt DIP socket for PLCC package.

Table 2. EPROM/Microcontroller GUPI Module Adapters

Device Type	GUPI 27010	GUPI 27011	GUPI 27210	GUPI 8742	GUPI MCS-51	GUPI 8796	GUPI 8796LCC
EPROM	27010	27011	27210				
Peripheral Microcontroller				8741AH 8742AH	87C51 8752BH 87C252	8794BH 8795BH 8796BH 8797BH	8796BH 8797BH
Package Types	DIP	DIP	DIP	DIP	PLCC DIP	PGA DIP	LCC

The iUP-Fast 27/K Personality Module

With the iUP-Fast 27/K personality module the user can program, read, and verify the contents of Intel's high density EPROMs, from the page-programmable (512K) 27513, to the CMOS 27C64, 27C256, and 87C64 EPROMs. This personality module supports the int$_e$ligent Programming algorithms and the int$_e$ligent Identifier™. The int$_e$ligent Identifier lets the personality module interrogate the PROM device in the program/master socket. It determines whether the type selected matches the type of PROM device installed and then selects the proper int$_e$ligent Programming algorithm. The int$_e$ligent Programming algorithms reduce programming time up to a factor of 10.

Low cost, plug-in upgrade kits allow addition of support for Intel's latest EPROMs. The first upgrade kit added support for the 27512 and innovative page-programmable 27513 plus the 27128A and 2817A. It has now been replaced by a second upgrade kit, iUP-Fast 27/K-U2 adding support for Intel's new CMOS EPROMs. (refer to Table 3).

As shown in Figure 3 the iUP-Fast 27/K personality module contains two 28-pin sockets, a hexadecimal display (0 through F), and a red LED that indicates when power is being applied to a socket. The program socket holds the device being programmed. The master socket will be used in future upgrades. The hexadecimal display shows the PROM device type selected.

Table 3. FAST27/K Module Device Support

Prom Type	Fast 27/K Module	Fast 27/K U2 Kit	Fast 27/K-CON* Kit
EPROM	2764	2764	2764
	2764A	2764A	2764A
		27C64	27C64
		87C64	87C64
	27128	27128	27128
		27128A	27128A
	27256	27256	27256
		27C256	27C256
		27512	27512
		27513	27513
E^{2}PROM		2817A	2817A

*Uses Quick-Pulse Programming Algorithm.

THE iPPS SOFTWARE

The iPPS software, included with the iUP-PC brings increased flexibility to PROM programming. The

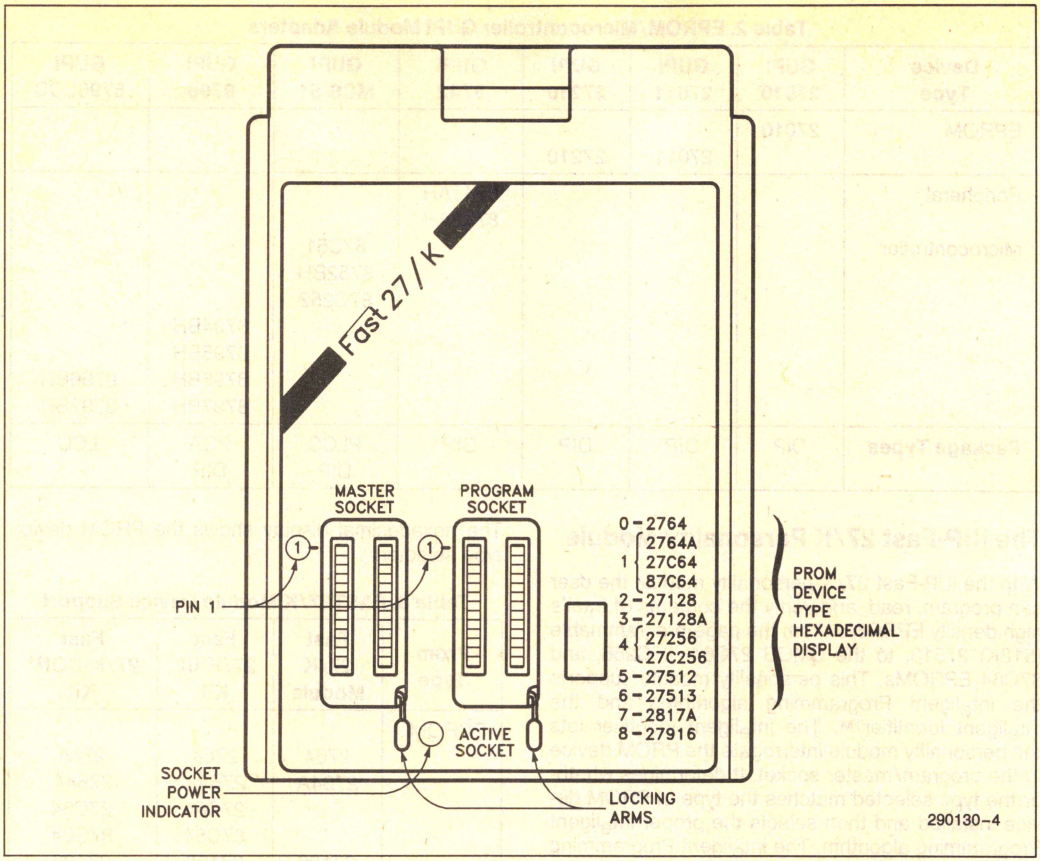


Figure 3. iUP-Fast 27/K Personality Module with U2 Upgrade

iPPS software provides user control through an easy-to-use interactive interface and performs the following functions to make programming quick and easy:

- Reads PROMs, ROMs and EPLDs.
- Programs PROMs directly or from a file.
- Verifies PROM data with buffer data.
- Prints PROM buffer, or device file contents on the system printer.
- Performs interactive formatting operations such as interleaving, nibble swapping, bit reversal, and block moves.
- Programs multiple PROMs from the source file, prompting the user to insert new PROMs.
- Uses a buffer to change PROM contents.

With the iPPS software the user can load programs from system memory or directly from a disk file. Access to the disk lets the user create and manipulate data in a virtual buffer. This block of data can be formatted into different PROM word sizes for program storage into several different PROM types. In addition, a program stored in the target PROM, the system memory, or a system disk file can be interleaved with a second program and entered into a specific target PROM or PROMs.

The iPPS software supports data manipulation in the following Intel formats: 8080 hexadecimal ASCII, 8080 absolute object, 8086 hexadecimal ASCII, 8086 absolute object, 80286 absolute object, and 80386 bootloadable object. Addresses and data can be displayed in binary, octal, decimal, or hexadecimal. The user can easily change default data formats as well as number bases.

iUP-PC SPECIFICATIONS

HOST SYSTEM

The iPPS will run on an IBM PC/XT, PC/AT or other true compatible with a DOS operating system. The PCPP requires one full-sized card slot inside the PC.

OPERATING ENVIRONMENT

Electrical Characteristics

PCPP:

Worst Case Power Consumption at IBM PC I/O Channel

Supply Voltage	Voltage Variance	Personality Module	Max. Current Drain
+5V	+5%, −4%	FAST27K	1.898 A
−12V	+10%, −9%	FAST27K	102.9 mA
+12V	+5%, −4%	GUPI	530 mA

Physical Characteristics

PCPP:

Length: 13.3 inches (33.9 cm)
Height: 3.9 inches (10.0 cm)

Interconnect Cable:

50 lead ribbon cable
Length: 3.0 feet (91.4 cm)
Width: 2.43 inches (5.5 cm)

iUP-GUPI:

Length: 7.0 inches (17.8 cm)
Width: 5.5 inches (1.4 cm)
Height: 1.6 inches (4.1 cm)

Environmental Characteristics

Environmental Class: B

Temperature:

Operating	10 to 40 degrees C
Non-Operating	−40 to 70 degrees C

Relative Humidity:

Operating	85% Maximum
Non-Operating	95% Maximum

DOCUMENTATION

168161—PCPP User's Guide

166428—iUP-GUPI Module User's Guide

User's Guides for Adaptors, FAST 27/K Modules, and upgrades included with respective units.

ORDERING INFORMATION

Order Code	Product Description
iUPPC	Universal Programmer for the Personal Computer: PCPP Programming Card, 50-Lead Interconnect Cable, iUP-GUPI, iPPS, PCPP User's Guide
ADAPT24TO28	28-Pin PLCC Socket Adapter for GUPI LOGIC-09 and GUPI LOGIC-IID
ADAPT40TO44	44-Pin PLCC Socket Adapter for GUPI LOGIC-09 and GUPI LOGIC-IID
piUPGUPI	Generic Universal Programmer Interface (Base)
GUPI LOGICIID	iUP-GUPI Logic Adapter
GUPI LOGIC09	iUP-GUPI Logic Adapter
GUPI LOGIC12	iUP-GUPI Logic Adapter
GUPI LOGIC18	iUP-GUPI Logic Adapter
GUPI LOGIC18PGA	iUP-GUPI Logic Adapter for 5C180 PGA
GUPI LOGICBIC	iUP-GUPI Logic Adapter
GUPI27010	iUP-GUPI EPROM Adapter
GUPI27011	iUP-GUPI EPROM Adapter
GUPI27210	iUP-GUPI EPROM Adapter
GUPI8742	iUP-GUPI Peripheral Adapter
GUPIMCS51	iUP-GUPI Microcontroller Adapter
GUPI8796	iUP-GUPI Microcontroller Adapter
GUPI8796LCC	iUP-GUPI Microcontroller Adapter
piUPFAST 27K	EPROM Personality Module
iUPFAST 27KU2	FAST 27/K Upgrade Kit
iUPFAST 27KCON	Adds Quick-Pulse algorithm and device support
iUPFAST 27KIT	Combines piUPFAST 27K and iUPFAST 27KU2

SCHEMA II-PLD

SCHEMATIC CAPTURE SOFTWARE

SCHEMA II-PLD is a powerful, low-cost schematic capture software package for designing with Intel EPLDs and with standard MSI, SSI, and discrete components. For EPLD designs, SCHEMA II-PLD outputs Advanced Design Files (ADFs) that can subsequently be compiled by iPLS II software. SCHEMA II-PLD supports EPLD symbols as well as MSI and SSI macro symbols, allowing designs to combine TTL and EPLD symbols as needed. The ability to create user-defined macro symbols that can be translated into ADF macro calls adds to SCHEMA II-PLD's power and versatility.

The EPLD Manager included in SCHEMA II-PLD provides a single user interface to both SCHEMA II-PLD and iPLS II software. EPLD Manager software is also available separately for users who already own SCHEMA II.

000275-1

iPLS II MACRO LIBRARIAN

The iPLS II (Intel Programmable Logic Software II) Macro Librarian is an optional software package that allows designers to create user-defined macro libraries for EPLD designs. User-defined macro functions are first developed as individual macro files using an ASCII text editor. These files are then combined into macro libraries by the Macro Librarian. Macro calls to use the functions can then be placed in iPLS II ADFs (Advanced Design Files) where they will be expanded during compilation by the iPLS II Macro Expander. Use of macros in iPLS II Advanced Design Files (ADFs) allows EPLD design to proceed at a higher level than with EPLD primitives alone.

Note that a preconfigured library of TTL macro functions is available from Intel to all registered iPLS II users. The Macro Librarian is not needed to use this TTL library. It is designed for users who need to create libraries that contain user-defined macros.

000275-2

UTILITIES

FUNCTIONAL SIMULATOR UTILITY

Description:

Simulation of EPLDs is supported with Intel's SIM (Functional Simulator). Combinatorial as well as registered designs can be simulated and circuit operation verified before a device is programmed. The design is simulated with a user generated vector file.

Availability:

SIM is a standalone utility that runs on any IBM PC, XT, AT or compatible. The simulator is available at no cost to Intel EPLD customers. Contact your local Intel sales office.

PAL2ADF UTILITY

Description

This document is a brief note on the use of the PAL2ADF program in translating PALASM 1 files into Intel's Advanced Design File (ADF) format. Descriptions for actual use can be found on the accompanying Manual page in the file PAL2ADF.MAN.

The PALASM file serves as a template for mapping the PALASM equations into ADF. The translation is performed as follows:

1) Read PAL description, and set the PAL pins to their appropriate EPLD primitive counterparts
2) Parse file and produce network description
3) Translate equations to ADF

PAL Configuration Database

When it is translating a PALASM file, PAL2ADF reads a database (default:PAL2ADF.DAT) that tells it:

- How many pins the PAL has
- Which default EPLD to translate to
- What pins are special inputs (Clock and Output Enable defaults)
- What EPLD I/O primitives to use for each PAL pin

The EPLD I/O primitives specify the network architecture that the EPLD must take on in order to mimic the functionality of the PAL. See the PAL2ADF.DAT file for more information.

Reconfiguring Outputs

In step (2) above, several checks are done in order to make sure that the network is configured appropriately. These primarily involve output pins, although input pins can be specified as well.

The first reconfiguration is for active low outputs in their equations. i.e.,

```
    PALASM: /SIGNAL = A * /B + C
becomes
        ADF: SIGNAL = /(A * /B + C);
```

The other reconfigurations are slightly more complex. Consider a PAL pin X which is an output with a D-latch. The output value is fed back into the P-term array after the Output Enable. This is described as a Registered Output Registered Feedback (RORF) in the Intel EPLDs. The default network description for this pin then is:

```
NETWORK:
    . . .
    X,X = RORF (Xp,CLK,GND,GND,OE)
    . . .
```

where CLK and OE are the default Clock and Output Enable signals.

Normally, there would be an equation that would describe Xp. (The 'p' is used to name the P-term value.) If, however, the X feedback is never used in an equation, then the I/O macrocell is reconfigured to a Registered Output No Feedback (RONF).

NETWORK:

...

X = RONF (Xp,CLK,GND,GND,OE)

...

For those I/O pins on the PAL which are used strictly as inputs, these use the Combinatorial Output I/O Feedback (COIF) primitive, with the Output Enable shut off (GND). The P-term is tied to the feedback, in order to satisfy the semantics of ADF.

NETWORK:

...

YY,YY = COIF (YYp,GND)

...

EQUATIONS:

...

YYp = YY;

If the PAL pin is being used strictly as an output and is never used in an equation, then the primitive is reconfigured to a Combinatorial Output No Feedback (CONF).

NETWORK:

...

YY,YY = COIF (YYp,GND)

...

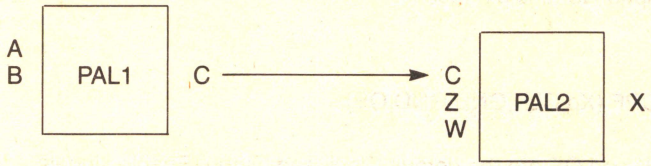
This is the same as above where a RORF is reconfigured to a RONF.

Multiple PAL Designs into 1 EPLD

It is possible to incorporate multiple PALASM descriptions into one EPLD. If each PALASM description is disjoint, (i.e., they have different pin names for each pin) then you can simply translate each file (with the pinlist information OFF) and compile them together with the iPLS Logic Optimizing Compiler (LOC).

The compiler allows you to specify multiple ADF files, allowing different subnetworks within one EPLD. You will probably want to use a larger EPLD to fit all the designs in.

If the PAL designs are not disjoint, then there are some steps that can be done by hand to integrate the designs. A simple example would be where one PAL feeds another a signal, and the second uses that to generate another signal.



In this case, C is an output of PAL1, and an input to PAL2. In PAL2, C,Z, and W generate the signal X. Suppose we have the equations:

```
PAL1
    /C = A * /B

PAL2
    X = /C*Z*W
        + C*/Z*W
```

In the resulting ADFs, the following NETWORKS are produced:

```
ADF for PAL1:
    NETWORK:
    A  = INP(A)
    B  = INP(B)
    C  = CONF(Cp,VCC)
    EQUATIONS:
       C = /(A * /B);

ADF for PAL2:
    NETWORK:
    Z  = INP(Z)
    W  = INP(W)
    C  = INP(C)
    X,X = RORF(Xp, CLK, GND, GND, OE)
    EQUATIONS:
    Xp = /C*Z*W
         + C*/Z*W;
```

These can be joined together into a single ADF:

```
    NETWORK:
    A  = INP(A)
    B  = INP(B)
    Z  = INP(Z)
    W  = INP(W)
    X,X = RORF (Xp, CLK, GND, GND, OE)
    EQUATIONS:
    C = /(A * /B);
    Xp = /C*Z*W
         + C*/Z*W;
```

Notice how C is now an intermediate variable rather than an actual signal. This is obviously a simple example, yet similar techniques can be applied to more complex cases. As much more logic can be placed into larger EPLDs, the job of splitting functions across multiple devices is reduced.

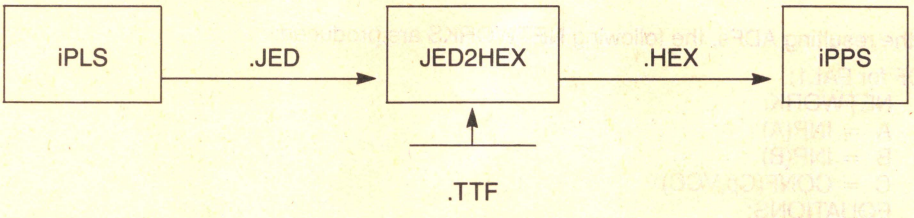
Availability

The PAL2ADF utility is available at no cost to Intel EPLD customers. Contact your local Intel sales office.

JED2HEX CONVERSION UTILITY

Description

JED2HEX is a utility to convert JEDEC files created by iPLS (.JED) into Intellec HEX files which can then be read by Intel's iPPS software. This allows programming of EPLDs via Intel's iUP200A/201A using a GUP I base and the appropriate adaptor (e.g. LOGIC-12). The following diagram represents a typical development cycle.



INSTALLATION: To install the utility and its device specific files, place the master disk in drive A: and invoke the JINSTALL.BAT batch file with the destination path for the utility and device files. Example:

A: JINSTALL C: MYPATH

When using JED2HEX, attach the package description letter when entering the device type. That is, enter 5C121D for a 5C121 ceramic DIP when prompted for the device type. Entering 5C121 will result in:

***ERROR: Device File Missing

To determine the packages supported in your JED2HEX software, examine all the .ttf extension files; it is the .ttf files which the device type command attempts to match.

When using iPPS, a file format of 8080 or 8086 must be specified when copying the JED2HEX generated HEX file to the buffer or directly into a device. If 8080 or 8086 is not specified, the default file format type of 80386 will be chosen and a "GENERAL ERROR — ILLEGAL FILE TYPE SPECIFIED" will result. An example of the proper COPY format:

PPS> COPY a: filename.HEX TO PROM 86

Availability

The JED2HEX Conversion Utility is available at no cost to Intel EPLD Customers. Contact your local Intel sales office.

APPLICATION NOTE

AP-279

October 1987

Implementing an EPLD Design Using Intel's Programmable Logic Development System

LAKSHMI JAYANTHI
DSO APPLICATIONS

NOTE

This design can also be developed on iPLS II (Version II of Intel's Programmable Logic Software). Some of the prompts or message may vary slightly, but the overall procedure is identical.

INTEL CORPORATION, 1986

Order Number 280310-002

OVERVIEW

Welcome to the fascinating world of ERASABLE PROGRAMMABLE LOGIC DEVICES (EPLDs) and Intel's Programmable Logic Development System (iPLDS). This application note has been written for the newcomer to Intel's devices and design tools. It has been designed as a step-by-step guide through the tools but should also prove useful as a reference document for the experienced logic designer.

By the end of this application note you will have designed/solved multiple logic problems and be in a position to implement solutions to many of the digital design challenges you face today. It is anticipated that this application note will be used in conjunction with Intel's iPLS software. To increase the usefulness of this application note, Intel will supply a PCB card for you to experiment on and a sample diskette (see Appendix E for details).

This application note is divided into the following three sections:

1. An introduction to Erasable Programmable Logic Devices (EPLD)
2. An introduction to Intel's Programmable Logic Development System (iPLDS)
3. Implementation of EPLD and iPLDS using detailed examples to implement a logic design.

INTRODUCTION

Programmable logic in the form of PALs have been available for some time. They have become more complex as Large Scale Integration (LSI) techniques have been applied to this technology.

The benefits of Large Scale Integration circuits are many fold. These circuits offer lower manufacturing costs, since the use of customized LSI circuits reduces required printed circuit board space, thereby significantly reducing board costs. These circuits also consume lower power so less expensive power supplies are required and cooling fans are also eliminated. LSI circuits also have higher reliability than equivalent systems comprised of many low density standard components.

As end users of semiconductors moved into higher and higher levels of integration, chip designers found it more and more difficult to define larger and larger blocks of logic. These difficulties led to the emergence of the user-defined Application Specific Integrated Circuit (ASIC).

The options available for application specific logic are explained below and shown in Figure 1.

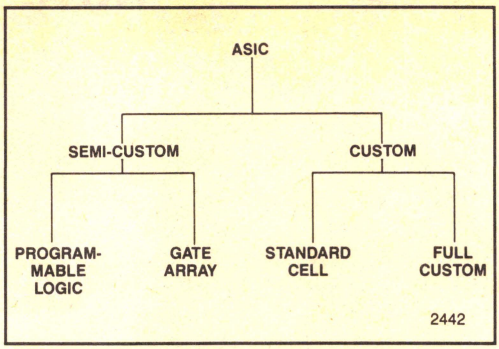


Figure 1. Logic Options

Full Custom: These circuits can be tailored to give the best functional performance with the highest level of integration, the smallest silicon area, the lowest power use, and be produced for the least cost at high production volumes.

Standard Cell Library: This approach represents an integrated circuit which is composed of predesigned and precharacterized cells chosen from a computer data base library of cells.

Gate Arrays: These are integrated circuits that contain a regular, usually square, matrix of predefined logic gates.

User Programmable Logic: The concept of user programmable logic is to provide the designer with the benefits of custom LSI chips from standard products.

A recent innovation in the programmable logic field has been Intel's introduction of an ERASABLE Programmable Logic Device. Using the same technology used in the manufacture of EPROMs, Intel now offers increased flexibility to the logic designer.

Intel has addressed the limitations of gate arrays and fuse programming logic with its EPLD products and development system support tools. The benefits to the system designer are:

- Greatly reduced lead times
- Low design costs
- Ease of design changes
- Low power dissipation from CHMOS technology
- Multiple programming facility
- Maximum flexibility in each chip and the ability to erase and reprogram
- High density products that maximize function, integration, and quality
- A self-contained, low-cost sophisticated development system based upon the industry standard IBM PC XT or AT.

Table 1. Intels EPLDs

EPLD	Gates	Pins	Dedicated Inputs	I/O
5C031	300	20	10	8
5C060	600	24	4	16
5C090	900	40	12	24
5C121	1200	40	13	24
5C180	1800	68	12	48

EPLDs are now a cost-effective solution to the problem of large scale logic integration. EPLDs are the simplest form of high density application-specific logic to implement. At present, the following logic devices are available from Intel as shown in Table 1.

Intel's EPLDs use the "Sum Of Products" architecture with programmable AND and fixed OR gates to drive a combinatorial or registered output. Each of the devices listed in Table 1 has different attributes and resources targeted at specific applications.

In general each device contains multiple sets of programmable MACROCELLS as shown in Figure 2.

Everything is programmable (and erasable if you need to make modifications). Product terms may be generated from any combination of input terms—any terms not used are considered a "don't-care" in the array. The output register is also programmable—you can choose D-type, Toggle, SR, or even JK FLIP-FLOPs; you can even choose no output register if you only require combinatorial outputs. The clock and output enables are also programmable.

Intel EPLD devices are available in many configurations to fit most applications. A complete listing of data sheet availability is covered in Appendix E.

DESIGN TECHNIQUES USING INTEL'S EPLDS

Designing with EPLDs is similar to designing with standard TTL logic circuits. The focus moves from "how can I configure this design with standard parts" to "what else could I replace using this EPLD". Remember, if you ever use all of an EPLDs resources you just move up the device chain to the next bigger component—all of the work you did is DIRECTLY PORTABLE to a larger device.

Any network, either combinatorial or registered, has an equivalent two level form. Any logic circuit consisting of AND, OR, NOR, NAND, XOR Logic can easily be converted into the corresponding truth table. Any Boolean expression, no matter how complex, may be written in Sum-Of-Products form. This Sum-Of-Products expression that has been derived from the truth table can be reduced until it has as few product terms as possible. This procedure can be repeated for any complex network.

Let us consider a very simple network as shown in Figure 3. This logic circuit consists of an AND gate, an OR gate and a NOT gate. The inputs are A, B, C, and the output is Y.

For this simple network, the truth table is shown in Table 2:

A Boolean expression can easily be written from the truth table in a Sum-Of-Products form. This expression contains the relationship between the inputs and the output.

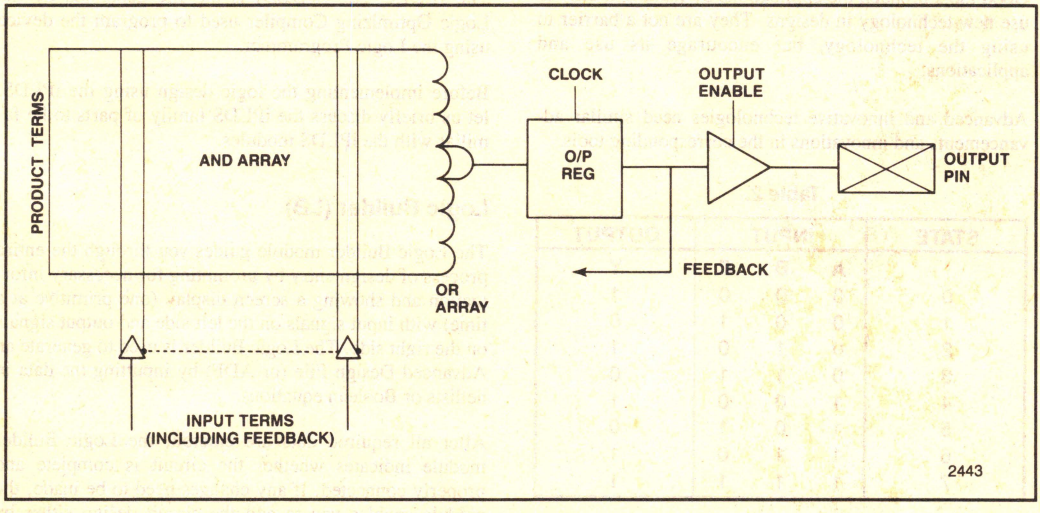


Figure 2. Macrocell Arch

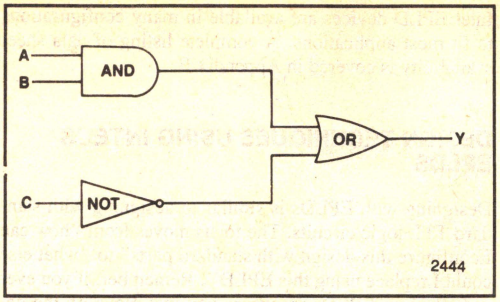


Figure 3. Simple Network

Note that the output Y is true in five of these eight states (0,2,4,6, and 7) so expressing Y in the form "Sum-Of-Products" by writing the ones in terms of A, B, and C yields:

$$Y = /A*/B*/C + /A*B*/C + A*/B*/C + A*B*/C + A*B*C$$

Hence, given any network, that network can be converted into its truth table. Next, a Sum-Of-Products expression that has the same truth table can be derived. If so desired, this Sum-Of-Products expression can be reduced using DeMorgan's theorem to simplify the circuit (you will see later that this will not be required).

DEVELOPMENT SUPPORT

Development tools are critical to the use of new technologies because tools allow you to control and use a new technology. Good tools help you, the designer, to work in familiar methods, then translate the design to the device.

Good tools broaden the applications by making it easy to use new technology in designs. They are not a barrier to using the technology, but encourage its use and applications.

Advanced and innovative technologies need similar advancements and innovations in the corresponding tools.

Table 2.

STATE	INPUT			OUTPUT
	A	B	C	Y
0	0	0	0	1
1	0	0	1	0
2	0	1	0	1
3	0	1	1	0
4	1	0	0	1
5	1	0	1	0
6	1	1	0	1
7	1	1	1	1

iPLDS, Intel's Programmable Logic Development System, provides a full spectrum of ways to design and use a variety of design tools with fast, easy-to-use entry software.

The iPLDS contains all the software, hardware, documentation and devices needed to program EPLDs. iPLDS are the most advanced PLD design tools available. It provides better utilization of device resources (more gates per chip) than any other development software. These versatile tools are for users with different skill levels and applications. iPLDS tools handle the details of converting your design to working silicon on the personal computer.

The iPLDS contains the three fundamental modules

- Logic Builder (LB)
- Logic Optimizimg Compiler (LOC)
- Logic Programmer Software (LPS)

To implement the logic design we will use the iPLDS modules in the order listed above.

The modules are essentially independant modules that use special data files to pass information as shown in Figure 4. These data files are the ADF, RPT, LEF, and JED files.

The Advanced Design File (*.ADF) is generated from the Logic Builder and contains the Inputs/outputs and all the primitive equations.

The Logic Equation File (*.LEF) contains the primitive equations that have been minimized by the Logic Optimizing Compiler.

The Utilization Report File (*.RPT) contains information on the macrocell and pin assignments.

The JEDEC File (*.JED) is the file generated by the Logic Optimizing Compiler used to program the device using the Logic Programmer.

Before implementing the logic design using the iPLDS, let us briefly discuss the iPLDS family of parts to be familiar with the iPLDS modules.

Logic Builder (LB)

The Logic Builder module guides you through the entire process of design entry by prompting for necessary information and showing a screen display (one primitive at a time) with input signals on the left side and output signals on the right side. The Logic Builder is used to generate an Advanced Design File (or ADF) by inputting the data in netlists or Boolean equations.

After all required data are entered, the Logic Builder module indicates whether the circuit is complete and properly connected. If any changes need to be made, the module enables you to edit the circuit design either by

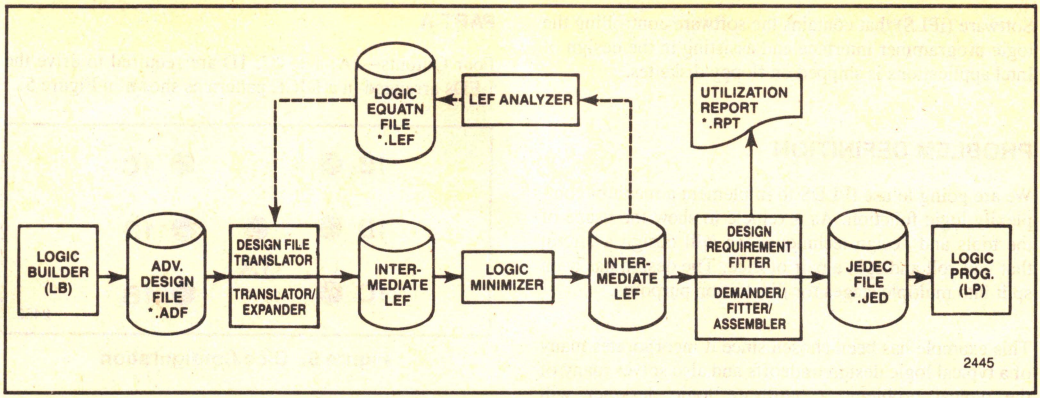


Figure 4. Block Diagram of iPLDS Modules

systematically scanning through the primitives in the Advanced Design File (ADF) or by directly finding a primitive by the name of a node connected to it.

Any circuit may be edited. The Logic Builder reads in the ADF and prompts you for changes. The Logic Builder also allows two or more partially complete ADF files to be MELDED together to form a more complex function. This concept is not discussed in this application note but will be a topic of a future application note.

Logic Optimizing Compiler (LOC)

The Logic Optimizing Compiler provides an easy-to-use interface to the Logic User System software. Regardless of the type of design entry method used, the LOC first translates an Advanced Design File (ADF) into internal logic equations; then it performs a Boolean reduction on the translated design, and finally produces a JEDEC Standard File, which is then used to program an Intel EPLD. In addition, you have the option of requesting an analysis of the Logic Equation File (LEF) as output by the Minimizer module.

The LOC performs the following functions:

- The TRANSLATOR translates the ADF into an intermediate Logic Equation File (LEF). (Most errors are detected and corrected).
- The EXPANDER expands the Boolean equations into Sum-Of-Products form, removes redundant factors from product terms, and produces another LEF.
- The MINIMIZER performs a sophisticated Boolean reduction on the translated design to maximize utilization of the EPLD.
- The LEF Analyzer converts the LEF output by the MINIMIZER into a human readable file to allow you to see your design. (*.LEF)
- The DEMANDER organizes the file output by the MINIMIZER.
- The FITTER matches your design requirements with the known resources of the Intel device.
- The ASSEMBLER converts the fitted requests into JEDEC file.

Logic Programmer Software (LPS)

The Logic Programmer Software provides a user interface to the JEDEC Standard File output of the Logic Optimizing Compiler and to the Logic Programmer Interface. You can use the Logic Programmer Software to view JEDEC files and to program your designs into EPLDs.

The Logic Programmer Software is used

- to program your designs into EPLDs
- to verify the validity of data in the device
- to read data from the device
- to display JEDEC data graphically
- to edit JEDEC data

HARDWARE REQUIREMENTS

The iPLDS requires an IBM PC XT, PC AT, or other compatible computer. A color monitor is preferred. The computer must have at least one 360K double-sided double-density disk drive, a second 360K floppy disk or hard disk, and at least 512K bytes of RAM memory.

The iPLDS consists of the Logic Programmer Interface card, and the programming unit needed to program and verify EPLDs. The Intel iUP 201 with a GUPI adapter may be used as an alternate system to program the EPLD devices.

SOFTWARE REQUIREMENTS

The personal computer should be capable of running DOS V3.0 or a higher version. The Intel Programmable Logic

Software (iPLS) that contains the software controlling the logic programmer interface and assisting in the design of Intel applications is shipped on floppy diskettes.

PROBLEM DEFINITION

We are going to use iPLDS to implement a medium complexity logic function. As a vehicle to show the usage of the tools and design techniques we will design a circuit that will roll and spin a pair of dice. The design has been split into multiple stages for illustration purposes.

This example has been chosen since it incorporates many of a typical logic design tradeoffs and also solves many of the typical problems a hardware logic designer will encounter.

Appendix A contains some basic definitions that may be useful when reading through the design and its implementation.

DESIGN SAMPLE

Problem Set-up

The circuit is designed to set both of the dice spinning when you push a switch and display a random set of numbers when you release the switch. The dice will spin at a rate that is visually pleasing and roll at the highest possible rate to ensure randomness.

You will implement the design in the following steps:

A. One dice that will roll out a number.

B. Add a switch that will control the roll/not roll action.

C. Add a second dice to roll a number.

D. Add a spinning option to both dice.

E. Retro-fit a power save feature to extend battery life.

Hence, at the end of the five design steps you will have a pair of dice spinning and showing a pair of numbers between 1 and 6 in a very random manner. At the end of the five design steps, you will have added a very realistic and practical feature to your design and that is extending the battery life by a power saving option. It is important to note that the five steps mentioned above are sequential steps in that step C can be achieved only after steps A and B etc. Let us describe the sample circuit for the dice rolling example. It is a very simple circuit allowing you to concentrate upon the design process. It illustrates the possible design stages and considerations in detail.

PART A

Four Outputs—1A, 1B, 1C, 1D are required to drive the LEDs arranged in a DICE pattern as shown in Figure 5.

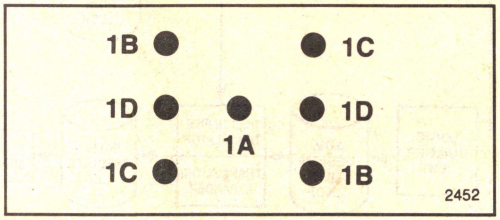


Figure 5. Dice Configuration

Operating sequence—Rolling dice from 1 to 6 and the block diagram of the circuit, both shown in Figure 6.

The total number of states that are possible is 16 since the four LED pairs generate a permutation of (2**4) = 16. The LEDs should be lit up such that any number between 1 and 6 inclusive is shown. Hence, out of the 16 possible states, only six states are valid. This leaves ten invalid states.

If the LEDs come up in a valid state upon power up, then a number between 1 and 6 will be displayed.

However, if the LEDs come up in an invalid state upon power up, then you have to design the circuit such that any one of the ten invalid states will fall into a valid state.

If the LEDs fall into any one of the ten invalid states, then you have designed the circuit to move into a state where 1A, 1B, 1C, 1D have zero logic values respectively on the next clock edge. Every time a zero logic value appears in the invalid states, then at the next clock edge, LED 1A gets lit up generating a valid state. Since 1 is a valid state, the numbers between 1 and 6 inclusive will be displayed at all subsequent clock edges.

Listed below are the steps involved in designing the logic circuit.

STEP 1. Generate the state diagram to clearly show the operating sequence including the status of the outputs for each state and the influence of the inputs on the next state transitions as shown in Figure 7. We have arbitrarily chosen that the states should count 1,2,3,4,5,6, and repeat. You could have implemented the design using any sequence but we chose the most obvious. Note how most of the invalid states move you to state 0 which then puts us into a valid state which then repeats forever.

STEP 2. Generate a truth table with entries for all available states and combinations of inputs, and use the next states resulting from these as shown in Table 3. The bracketed numbers, (3) etc., show the number being

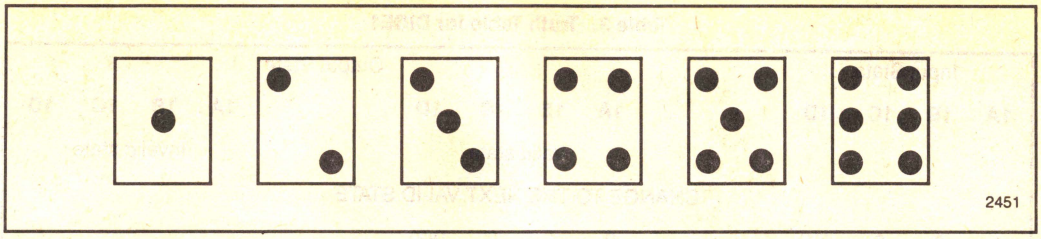


Figure 6. Rolling Sequence

displayed on the dice and the 0, 1 values of 1D, 1C, 1B, and 1A indicate which LEDs should be OFF/ON to display the required dice pattern.

STEP 3. Convert the truth table directly into Sum-Of-Products equations as shown below:

DICE1A has four entries; 3 from the valid states and one to control the invalid states

```
DICE1A = (/1A*1B*/1C*/1D + /1A*1B*1C*/1D
        + /1A*1B*1C*1D + /1A*/1B*/1C*/1D)
```

DICE1B has five entries from valid states

```
DICE1B = (1A*/1B*/1C*/1D + /1A*1B*/1C*/1D
        + 1A*1B*/1C*/1D + /1A*1B*1C*/1D
        + 1A*1B*1C*/1D)
```

DICE1C has three entries from valid states

```
DICE1C = (1A*1B*/1C*/1D + /1A*1B*1C*/1D
        + 1A*1B*1C*/1D)
```

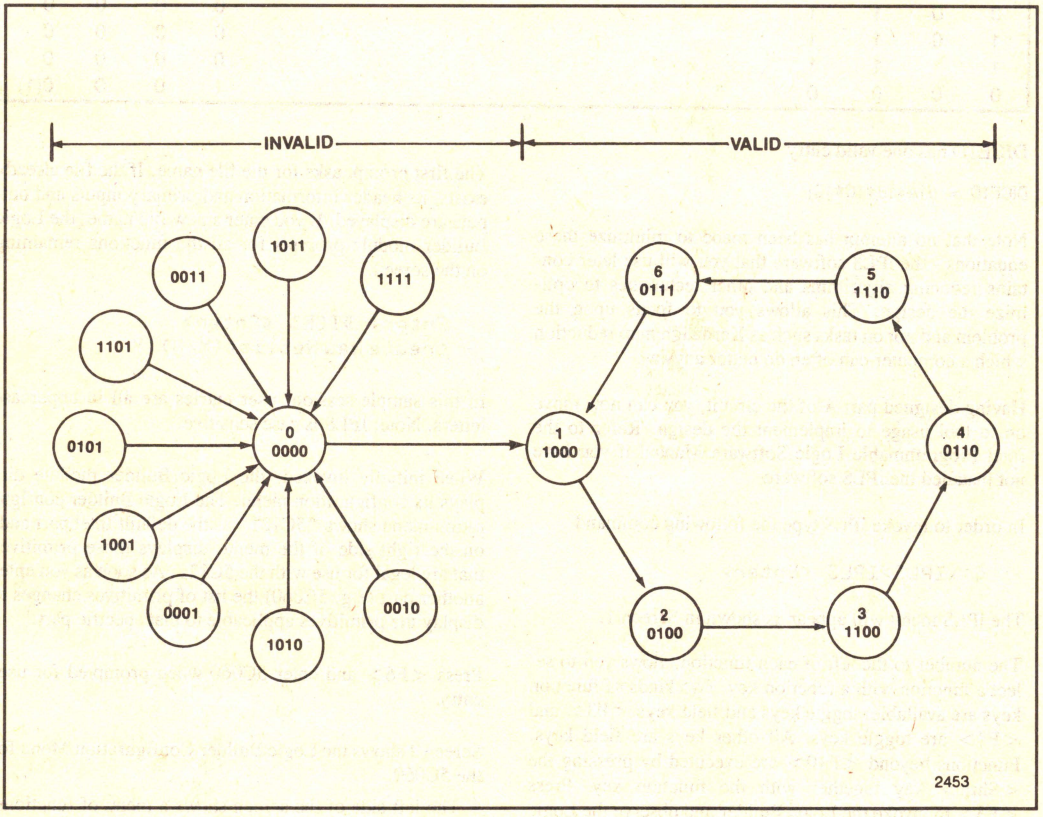


Figure 7.

Table 3. Truth Table for DICE1

Input State				Output State							
1A	1B	1C	1D	1A	1B	1C	1D	1A	1B	1C	1D
				Valid state				Invalid state			
CHANGE TO THE NEXT VALID STATE											
1	0	0	0(1)	0	1	0	0(2)				
0	1	0	0(2)	1	1	0	0(3)				
1	1	0	0(3)	0	1	1	0(4)				
0	1	1	0(4)	1	1	1	0(5)				
1	1	1	0(5)	0	1	1	1(6)				
0	1	1	1(6)	1	0	0	0(1)				
CONTROL THE INVALID STATES											
0	0	1	0					0	0	0	0
1	0	1	0					0	0	0	0
0	0	0	1					0	0	0	0
1	0	0	1					0	0	0	0
0	1	0	1					0	0	0	0
1	1	0	1					0	0	0	0
0	0	1	1					0	0	0	0
1	0	1	1					0	0	0	0
1	1	1	1					0	0	0	0
0	0	0	0					1	0	0	0(1)

DICE1D has one valid entry

```
DICE1D = (/1A*1B*1C*1D)
```

Note that no attempt has been made to minimize these equations - the iPLS software that you will use later contains reducing algorithms and other techniques to optimize the design. This allows you to focus upon the problem and not on tasks such as Karnaugh map reduction which a computer can often do better anyway.

Having designed part A of the circuit, you can now move on to tool usage to implement the design. Refer to the Intel Programmable Logic Software Manual if you have not installed the iPLS software.

In order to invoke iPLS type the following command

```
C:\IPLS>IPLS <Enter>
```

The iPLS menu will appear as shown in Screen 1.

The number to the left of each function allows you to select a function with a function key. Two kinds of function keys are available: toggle keys and field keys. <F3> and <F4> are toggle keys. All other keys are field keys. Functions beyond <F10> are executed by pressing the <Shift> key together with the function key. Press <F3> to invoke the Logic Builder and observe the Logic Builder menu as shown in Screen 2.

The first prompt asks for the file name. If the file already exists, its header information and primary inputs and outputs are displayed. If you enter a new file name, the Logic Builder module prompts for all the functions remaining on the screen.

```
Enter: DICE1 <Enter>
Create New Netlist(Y/N):Y
```

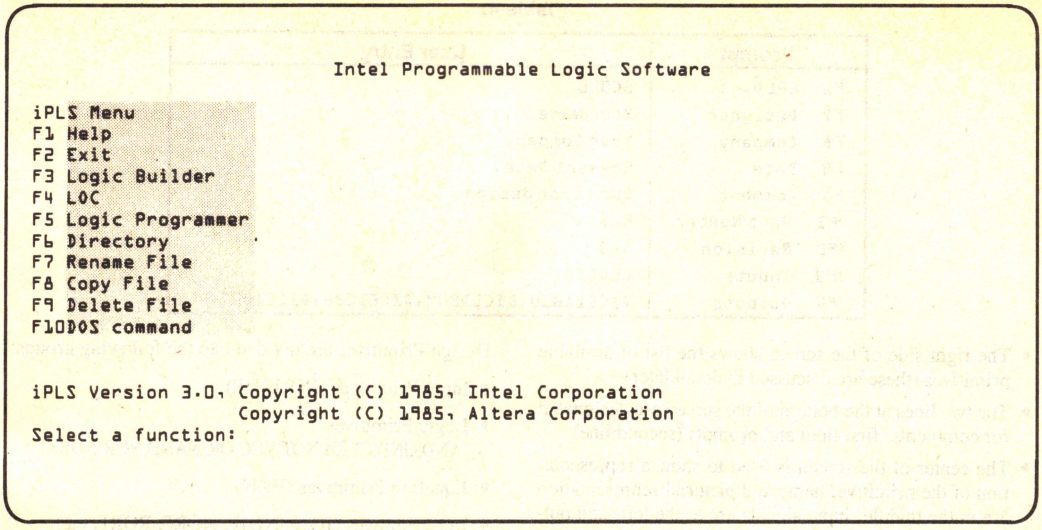
In this sample session, user entries are all in uppercase letters. Note: IPLS is case sensitive.

When initially invoked, the Logic Builder module displays its configuration menu. The Logic Builder configuration menu shows "5C121" as the default Intel part and, on the right side of the menu, displays those primitives that are legal for use with the 5C121. As soon as you enter another part (e.g. 5C060) the list of primitives changes to display the primitives applicable to that specific part.

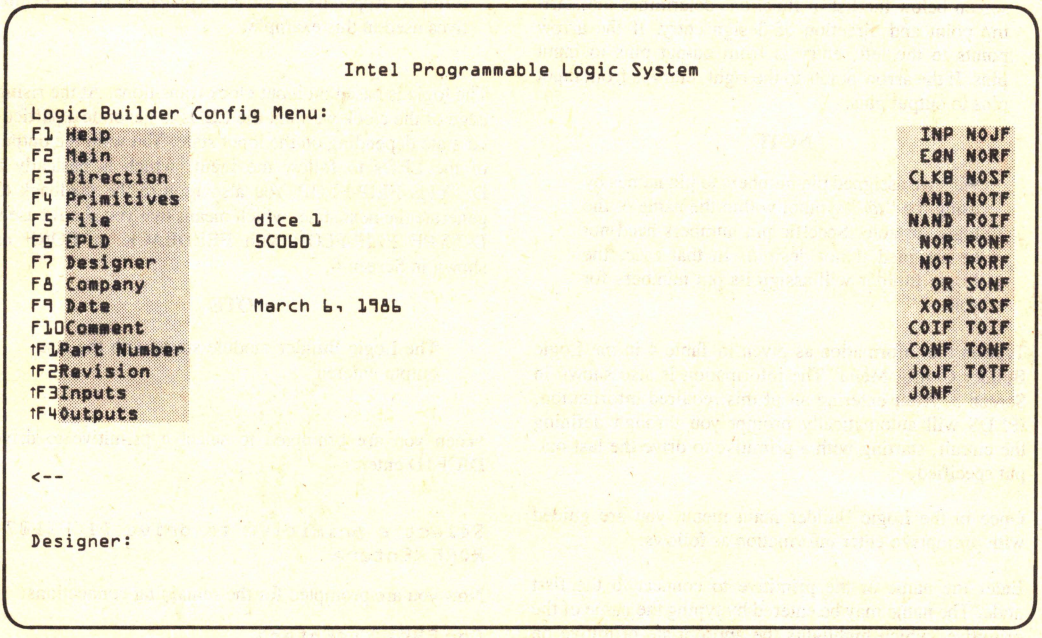
Press <F6> and enter 5C060 when prompted for user entry.

Screen 2 shows the Logic Builder Configuration Menu for the 5C060.

- The left side of the screen shows a menu of functions, each preceded by a function key number.



Screen 1.



Screen 2.

Table 4.

Prompt	User Entry
F6 EPLD	5C060
F7 Designer	Your Name
F8 Company	Your Company
F9 Date	Present Date
F10 Comment	Our first design
↑F1 Part Number	0.1
↑F2 Revision	1.0
↑F3 Inputs	CLOCK@1
↑F4 Outputs	DICE1A@10,DICE1B@9,DICE1C@8,DICE1D@7

- The right side of the screen shows the list of available primitives (these are discussed in detail later).
- The two lines at the bottom of the screen are designated for comments (first line) and prompts (second line).
- The center of the screen is used to show a representation of the primitive; name and pictorial representation are in the middle, input signals are to the left, and output signals are to the right of the primitive.
- The direction of the arrow located on the left side of the screen below the list of functions determines the starting point and direction of design entry. If the arrow points to the left, entry is from output pins to input pins. If the arrow points to the right, entry is from input pins to output pins.

NOTE

We have assigned pin numbers to pin names by using the "@" symbol within the name of the logic variable. Specific pin numbers need not be assigned if not desired. In that case, the Logic Builder will assign its pin numbers for you.

Type in the information as given in Table 4 in the Logic Builder Config Menu. The information is also shown in Screen 3. After entering all of this required information, iPLDS will automatically prompt you through defining the circuit, starting with a primitive to drive the last output specified.

Once in the Logic Builder main menu, you are guided with prompts to enter information as follows:

Enter the name of the primitive to connect to the first node. The name may be entered by typing the name of the primitive, which highlights the appropriate primitive on the right side of the menu, then pressing <Enter>.

Subsequently, a representation of the primitive is displayed in the center of the screen surrounded by input and output signals. You are prompted for names of nodes to connect to each of the signals. The Design Primitives library contains approximately 80 basic functional blocks needed for designing circuits in programmable logic products.

Design Primitives are divided into the following groups:

- Input Primitives (INP,LINP)
- Logic Primitives (AND,GND,CLKB,NOT,VCC,OR,NAND,NOR,XOR)
- Equation Primitives (EQN)
- I/O Primitives (JOJF, NOJF, NORF, RORF, etc)

Refer to Appendix A for an explanation of the Primitives used in this example.

The logic is based on input clock transitions. At the rising edge of the clock we want the LEDs to generate a particular state depending on the input state. You want the output of the LEDs to follow the input, which is basically a D-TYPE FLIP-FLOP. You also require the feedback to generate the next state, which means that you should use a D-TYPE FLIP-FLOP with FEEDBACK or RORF as shown in Screen 4.

NOTE

The Logic Builder module starts with the last output entered.

When you are prompted to select a primitive to drive DICE1D enter:

```
Select a primitive to drive DICE1D@7:
RORF <Enter>
```

Now you are prompted for the remaining connections:

```
For FBK: 1D <Enter>

For OE, P, C: Press <Enter> (VCC, GND are
the defaults).

For D: IN1D <Enter>

For CLK: CLOCK <Enter>

Select a primitive to drive CLOCK: INP
<Enter>
```

```
                        Intel Programmable Logic System

Logic Builder Config Menu:
F1 Help                                                          INP NOJF
F2 Main                                                          EQN NORF
F3 Direction                                                    CLKB NOSF
F4 Primitives                                                    AND NOTF
F5 File              dice 1                                     NAND ROIF
F6 EPLD              5C060                                       NOR RONF
F7 Designer          Your name                                   NOT RORF
F8 Company           Your company                                 OR SONF
F9 Date              March 6, 1986                               XOR SOSF
F10Comment           Our first design                           COIF TOIF
↑F1 Part Number      0.1                                        CONF TONF
↑F2 Revision         1.0                                        JOJF TOTF
↑F3 Inputs           clock@1                                    JOJF
↑F4 Outputs          DICE1A@10,DICE1B@9,DICE1C@8,DICE1D@7

<--

Outputs:DICE1A@10,DICE1B@9,DICE1C@8,DICE1D@7
```

Screen 3.

```
                        Intel Programmable Logic System

Logic Builder Main Menu:
F1 Help                                                          INP NOJF
F2 Exit                                                          EQN NORF
F3 New                  Oe                                      CLKB NOSF
F4 Open                 P                                        AND NOTF
F5 Find                 C                                       NAND ROIF
F6 Edit                 D                Out diceld              NOR RONF
F7 Config          Clk  ->               Fbk                     NOT RORF
F8 Node List                                                     OR SONF
F9 Redraw                  RORF                                 XOR SOSF
                                                               COIF TOIF
                                                               CONF TONF
                                                               JOJF TOTF
                                                               JONF

<--
Pin=7

Fbk:1d
```

Screen 4.

```
In: CLOCK <Enter>

Select a primitive to drive IN1D: EQN
<Enter>
```

After you are prompted for the equation, type it in as derived in the Problem Set-up section. Please note that "/" indicates a logical "NOT", "*" indicates a logical "AND", and "+" indicates a logical "OR". The equation is terminated by a ";" as shown in Screen 5.

```
IN1D = (1A * 1B * 1C * /1D); <Enter>
```

The following prompts and design entries, as shown in Table 5, are needed to complete the design entries for DICE1C, DICE1B, and DICE1A respectively.

The Logic Builder will stop prompting for primitives once you have entered the complete design.

Press <F8> to show the design so far as shown in Screen 6.

Press <F2> to exit.

The Logic Builder main menu is cleared, replaced by the Logic Builder exit menu.

To save the configuration and return to iPLS menu you must press <F6> (Save-Exit).

Note that you are saving the Advanced Design File (ADF) that is generated by the Logic Builder.

You can print the ADF file that has been created at the end of this session if you so desire. You can use <F10> when in the iPLS main menu to print the ADF file for a listing. You can verify your file with the DICE1.ADF file given in Appendix D. If you desire a listing, while you are in the iPLS main menu, type the following:

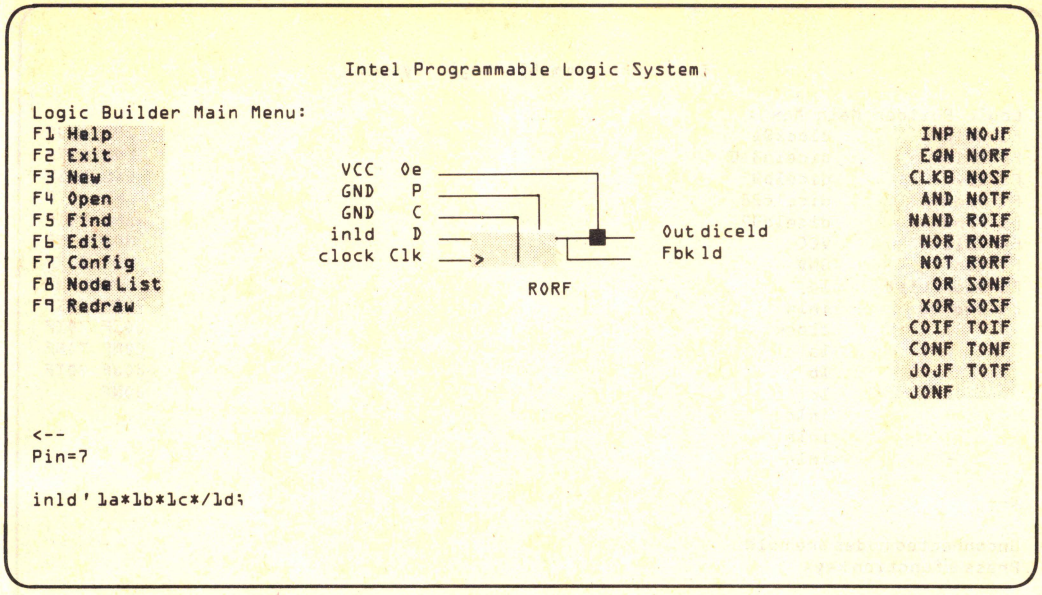
```
<F10> <Enter>

PRINT DICE1.ADF <Enter>
```

Submitting the ADF to the LOC

This ADF file is now compiled using the Logic Optimizing Compiler. To enter the ADF created with the Logic Builder module into the Logic Optimizing Compiler (LOC), press <F4> to access the LOC menu.

Table 5.

PROMPT	USER ENTRY
Select a primitive to drive 1C:	RORF <Enter>
Out:	DICE1C <Enter>
Oe:	VCC<Enter>
P:	GND <Enter>
C:	GND <Enter>
D:	IN1C <Enter>
Select a primitive to drive IN1C:	EQN <Enter>
IN1C:	(1A*1B*/1C*/1D)+(/1A*1B*1C*/1D)+(1A*1B*1C*/1D); <Enter>
Select a primitive to drive 1B:	RORF <Enter>
Out:	DICE1B <Enter>
Oe:	VCC <Enter>
P:	GND <Enter>
C:	GND <Enter>
D:	IN1B <Enter>
Select a primitive to drive IN1B:	EQN <Enter>
IN1B:	(1A*/1B*/1C*/1D)+(/1A*1B*/1C*/1D)+(1A*1B*/1C*/1D) +(/1A*1B*1C*/1D)+(1A*1B*1C*/1D);<Enter>
Select a primitive to drive 1A:	RORF <Enter>
Out:	DICE1A <Enter>
Oe:	VCC <Enter>
P:	GND <Enter>
C:	GND <Enter>
D:	IN1A <Enter>
Select a primitive to drive IN1A:	EQN <Enter>
IN1A:	(/1A*1B*/1C*/1D)+(/1A*1B*1C*/1D)+(/1A*1B*1C*1D) +(/1A*/1B*/1C*/1D);<Enter>



Screen 5.

Once the LOC menu is displayed, you are prompted through the LOC menu functions as follows:

The Input Format prompts you to specify your form of input: If input is in the form of a pinlist as output by DASH-2, enter P, if input is an Advanced Design File, enter an ADF or press <Enter> (ADF is the default). If output is a component list from PCAD, enter C.

```
INPUT FORMAT: A <Enter>

FILE NAME: DICE1 <Enter>

MINIMIZATION: <Enter to select default>

INVERSION CONTROL: <Enter to select default>

LEF ANALYSIS: <Enter to select default>
```

After you have answered all the prompts, you are asked if you wish to run under the above conditions as shown in Screen 7.

```
DO YOU WISH TO RUN UNDER THE ABOVE CONDITIONS [Y/N]?
```

Enter: Y

Finally you are prompted with:

```
WOULD YOU LIKE TO IMPLEMENT ANOTHER DESIGN [Y/N]?
```

Enter: N

Note that the LOC generates a synopsis of its progress as shown in Screen 8. You are returned to the iPLS menu.

At the end of the LOC a JEDEC Standard File has been created which we will use in the Logic Programmer, DICE1.JED.

Also at the end of the LOC a report file is created, DICE1.RPT, which gives the pin configuration menu of the device. The DICE1.RPT file is given in Appendix D.

Programming the EPLD

Finally, you submit your design to the Logic Programmer. In order for you to use the Logic Programmer, you must have the programming card plugged in. Please refer to the Intel Programmable Logic Software User Manual for installation instructions.

Alternatively you can use Intel's GUPI (Generic Universal Programmer Interface) to program your device.

```
                    Intel Programmable Logic System

Logic Builder Main Menu:
F1 Help          clock@1                                    INP NOJF
F2 Exit          dice1a@10                                  EQN NORF
F3 New           dice1b@9                                  CLKB NOSF
F4 Open          dice1c@8                                   AND NOTF
F5 Find          dice1d@7                                  NAND ROIF
F6 Edit          VCC                                        NOR RONF
F7 Config        GND                                        NOT RORF
F8 Node List     1d                                          OR SONF
F9 Redraw        in1d                                       XOR SOSF
                 clock                                     COIF TOIF
                 1a                                        CONF TONF
                 1b                                        JOJF TOTF
                 1c                                        JONF
                 in1c
                 in1b
                 in1a

<--

Unconnected nodes are bold
Press a function key:
```

Screen 6.

```
                    Intel Programmable Logic System

LOC Menu
F1 Help
F2 iPLS Menu
F3 Input Format        ADF
F4 File Name           dice1
F5 Minimization        Yes
F6 Inversion Control   No
F7 LEF Analysis        Yes


Do you wish to run under the above conditions [Y/N]?
```

Screen 7.

The iUP-GUPI and assorted GUPI LOGIC adaptors provide an alternative programming solution for Intel's H-series and EPLD devices, when purchased with the iPLS. This complete set of software is available without the Logic Programmer pod and the IBM interface card.

While you are still in the iPLS menu, press <F5>. This function allows you to access the Logic Programmer Software. The Logic Programmer will now come up as shown in Screen 9.

```
                    Intel Programmable Logic Software

LOC Menu
F1 Help                 ADF Minimization LEF-Analysis
F2 iPLS Menu            dice1
F3 Input Format
F4 File Name            ***INFO-LOC-Begin execution
F5 Minimization         ***INFO-LOC-ADF converted to LEF
F6 Inversion Control    ***INFO-LOC-S.O.P. LEF produced
F7 LEF Analysis         ***INFO-LOC-LEF reduced
                        ***INFO-LOC-LEF analyzed
                        ***INFO-LOC-Resource demand determined
                        ***INFO-LOC-Design fitting complete
                        ***INFO-LOC-JEDEC file output

                        LOC cycle successfully completed


Would you like to implement another design [Y/N]?
```

Screen 8.

Use the cursor keys to select "Program Device" option.

When you are prompted

Enter JEDEC file name

Enter: DICE1.JED <Enter>

When you are prompted for:

Select Device For Programming

Enter: 5C060 <Enter>

When you are prompted for:

Do you wish to enable verify protection? [Y/N]?

Enter: N

When you are prompted for:

Do you wish to enable turbo-bit? [Y/N]?

Enter: N

Once you have answered all the prompts, the device is programmed and ready to be used in an actual circuit, as shown in Screen 10.

Exit from the Logic Programmer after saving the JEDEC file by using the "EXIT" option.

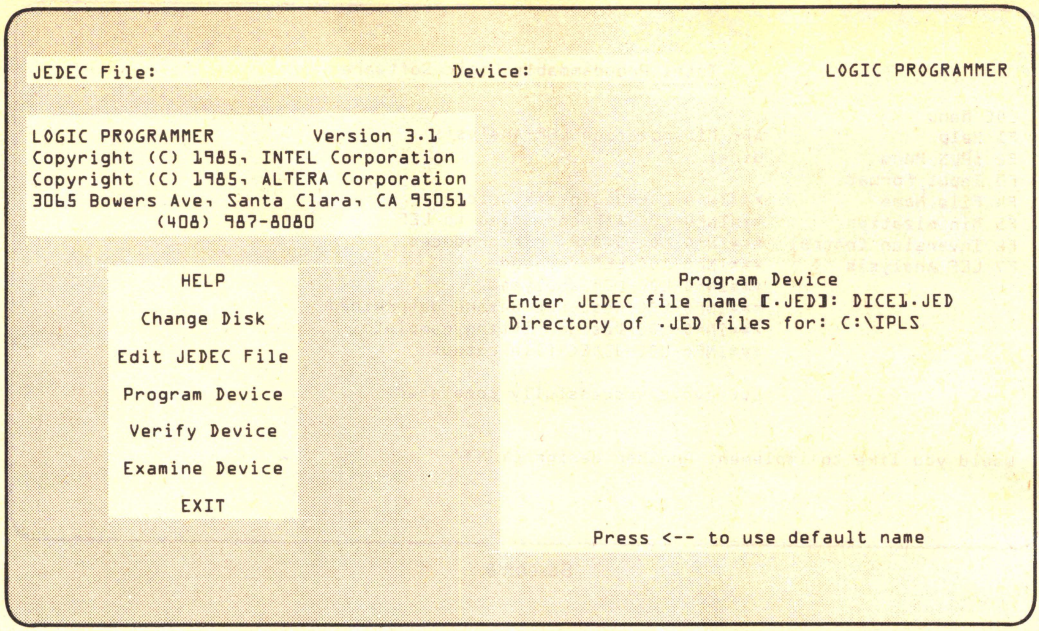
This completes part A of the design, which was to roll a single dice. The programmed device can be tested as described in Appendix C.

PART B

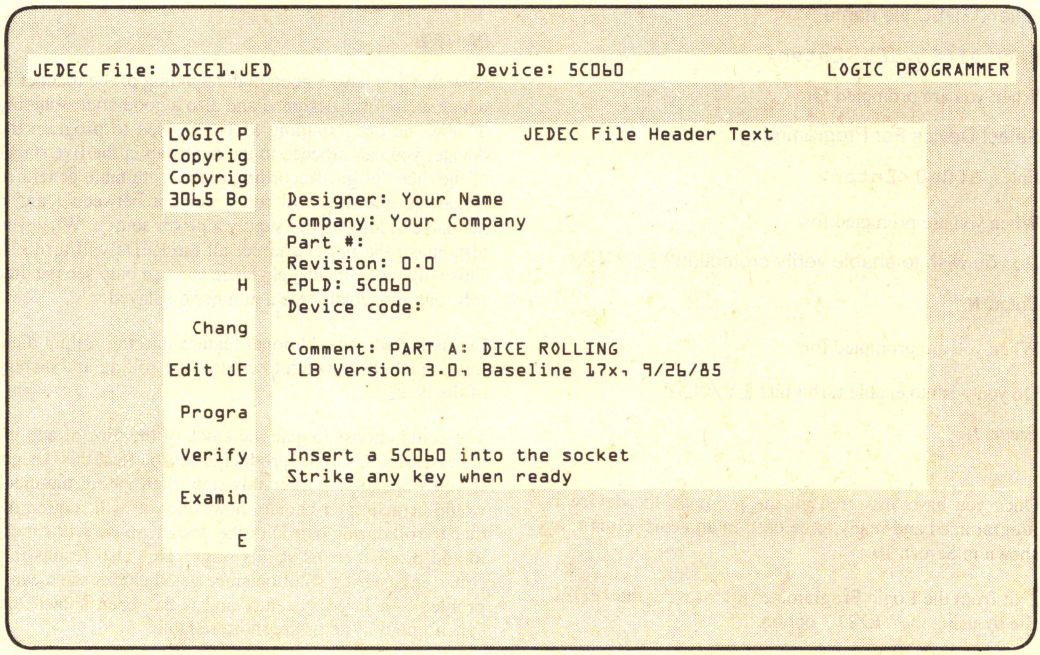
Now that you have a good understanding of the manner in which a circuit is designed and also a good understanding of how the programming tools are used to program the device, you can proceed to the next step in the five stages of the dice design. According to the truth table generated in part A, the dice will roll a number between 1 and 6 inclusive as long as you supply a power source. When you disconnect the power source, all the LEDs will turn off. This will not be much help since you can only see the dice roll, but not actually see a number displayed.

Let us include an additional feature into the rolling dice. Let us include a switch to control the rolling and display of the dice.

You could choose to gate the clock of the dice or add the necessary inputs to the product terms to effect this design. If you were to stop after this step, then gating the clock would be a simpler choice, however, you will require the dice to roll during part D of the design; so we will choose to add product terms at this stage. This also results in a better engineering solution since gated clocks often cause problems in large systems, and it has been shown that synchronous systems are more reliable.



Screen 9.



Screen 10.

Since you already have a proven design of a rolling dice from part A, we shall use the Logic Builder and edit that design. You may wish to save the original design at this stage. You can do this by using the <F10> key in the Main Menu. Press <F10> and issue the following command before re-entering the iPLS menu:

```
COPY DICE1.* DICE1A.*
```

The truth table is shown in Table 6.

Now you can use the iPLDS to design and program the device.

Go through the same steps to program the device as in Part A of the design example. Use the Logic builder, the Logic Optimizing Compiler, and the Logic Programmer respectively. The Logic Optimizing Compiler and the Logic Programmer steps are identical to the corresponding steps explained in part A of the design example. However, the Logic Builder will be used to edit the existing file, DICE1, to include the switch feature as follows:

Invoke the Logic Builder Menu from the iPLS main menu by pressing the <F3> key. Once you obtain the Logic Builder Configuration Menu, type in DICE1 as your input file name.

Use (Shift)(F3) to get the Inputs option and then add switch at pin #2 to it.

Inputs: CLOCK, SWITCH@2 <Enter>

Now press <F2> to exit to the Logic Builder Main Menu and answer the prompts as given in Table 7.

All that is left to do now is to edit the four equations, IN1A, IN1B, IN1C, IN1D to add the SWITCH option to it. Edit the four equations as follows:

Edit Function

When you press the "Edit" function key, <F6>, while in the main menu, the edit menu is displayed on the left side of the screen as shown in Screen 11. If you wish to edit an EQN Primitive displayed on the screen, press <F6>. Then the equation is moved to the prompt line where it can be edited.

Hence, the Boolean expressions for this case would consider the situations of when the switch was ON as well as OFF. The Boolean equations would contain the expression for the switch as follows.

```
DICE1A = ((1A*/1B*/1C*/1D)+(1A**1B*/1C*/1D)
         +(1A*1B*1C*/1D)
         +(/1A*/1B*/1C*/1D))*/SWITCH
         +((/1A*1B*/1C*/1D)+(/1A*1B*1C*/1D)
         +(/1A*1B*1C*1D)
         +(/1A*/1B*/1C*/1D))*SWITCH

DICE1B = ((/1A*1B*/1C*/1D)+(1A*1B*/1C*/1D)
         +(/1A*1B*1C*/1D)+(1A*1B*1C*/1D)
         +(/1A*1B*1C*1D))*/SWITCH
         +((1A*/1B*/1C*/1D)
         +(/1A*1B*/1C*/1D)+(1A*1B*/1C*/1D)
         +(/1A*1B*1C*/1D)
         +(1A*1B*1C*/1D))*SWITCH

DICE1C = ((/1A*1B*1C*/1D)
         +(1A*1B*1C*/1D)
         +(/1A*1B*1C*1D))*/SWITCH
         +((1A*1B*/1C*/1D)+(/1A*1B*1C*/1D)
         +(1A*1B*1C*/1D))*SWITCH

DICE1D = (/1A*1B*1C*1D)*/SWITCH
         +(1A*1B*1C*/1D)*SWITCH
```

The equation primitive must be displayed on the screen in order to edit that equation. In order to display the equation on the screen, use the "Find" command, <F5>, to find it.

The "Find" command prompts for a node name: then searches the design for that node and displays it. If the direction arrow points to the left, the primitive on the output side of the node is shown. If the direction arrow points to the right, the first primitive on the input side is shown.

After you have modified all four equations to include the SWITCH feature, return to the iPLDS main menu using the <F5> key and save the design using the <F6> key. You can verify your ADF file with the ADF file for part B given in Appendix D.

The file is ready to be compiled using the LOC, and the device is ready to be programmed using the LP.

The steps required to use the LOC and the LP are identical to the steps in part A.

Now the device that has been programmed is ready to be tested. At this stage in the design, you have completed part B of the design which is to add a switch to give the roll/no-roll option.

The programmed device can be tested as described in Appendix C.

Let us summarize before moving on to the next part of the design.

Table 6. Truth Table for DICE1

	Input State				Output State							
SWITCH	**1A**	**1B**	**1C**	**1D**	**1A**	**1B**	**1C**	**1D**	**1A**	**1B**	**1C**	**1D**
					Valid state				Invalid state			
					REMAIN IN THE SAME STATE							
0	1	0	0	0	1	0	0	0(1)				
0	0	1	0	0	0	1	0	0(2)				
0	1	1	0	0	1	1	0	0(3)				
0	0	1	1	0	0	1	1	0(4)				
0	1	1	1	0	1	1	1	0(5)				
0	0	1	1	1	0	1	1	1(6)				
					CONTROL THE INVALID STATES							
0	0	0	1	0					0	0	0	0
0	1	0	1	0					0	0	0	0
0	0	0	0	1					0	0	0	0
0	1	0	0	1					0	0	0	0
0	0	1	0	1					0	0	0	0
0	1	1	0	1					0	0	0	0
0	0	0	1	1					0	0	0	0
0	1	1	1	1					0	0	0	0
0	0	0	0	0					1	0	0	0(1)
					CHANGE TO THE NEXT VALID STATE*							
1	1	0	0	0(1)	0	1	0	0(2)				
1	0	1	0	0(2)	1	1	0	0(3)				
1	1	1	0	0(3)	0	1	1	0(4)				
1	0	1	1	0(4)	1	1	1	0(5)				
1	1	1	1	0(5)	0	1	1	1(6)				
1	0	1	1	1(6)	1	0	0	0(1)				
					CONTROL THE INVALID STATES							
1	0	0	1	0				0	0	0	0	
1	1	0	1	0				0	0	0	0	
1	0	0	0	1				0	0	0	0	
1	1	0	0	1				0	0	0	0	
1	0	1	0	1				0	0	0	0	
1	1	1	0	1				0	0	0	0	
1	0	0	1	1				0	0	0	0	
1	1	0	1	1				0	0	0	0	
1	1	1	1	1				0	0	0	0	
1	0	0	0	0				1	0	0	0(1)	

Note: This part of the truth table is identical to Table 3.

We have briefly discussed the EPLD and the IPLDS family of parts. We have also defined the design problem. We have implemented the design using the state equations and the truth table, edited an existing design to add features, and actually programmed a device using the Logic Builder, Logic Optimizing Compiler, and the Logic Programmer.

Our logic in implementing the dice example is to use the LED pairs in outputs 1A, 1B, 1C, and 1D respectively as

Table 7.

Prompts	User Entry
Select a primitive for switch@2 to drive:	INP <Enter>
Out:	SWITCH<Enter>
Select a primitive for switch to drive:	EQN<Enter>

shown in Figure 8. These LEDs are lit up to generate numbers between 1 and 6 inclusive. We are using a D-TYPE FLIP-FLOP to implement the truth table. The clock is a free running clock. A push button switch is also supplied to give the roll/no-roll option. Whenever the switch is ON, the LEDs roll, and when the switch is OFF, the LEDs display a number between 1 and 6, as long as the clock is supplied to the device.

After seeing the dice roll and display a number, you can either quit or move onto parts C, D and E of the design process. The following three parts describe a versatile use of the EPLD concept.

PART C

We are using an EPLD 5C060 which is a 24 pin, 600 gate device. It has four dedicated input pins and 16 input/output pins. Up to this point you have used only one input pin which is the switch and only four input/output pins for the four LEDs 1A, 1B, 1C, 1D.

Part C of the design is to include a second dice with the first dice. This is a step towards real-world application since dice are usually rolled in pairs. At the end of this section, you will have a pair of dice rolling and displaying a pair of numbers. All the conditions and truth tables and Boolean expressions that were designed for part B, hold good for DICE1. The equations for DICE2 would change slightly as explained below.

You have designed a 6 state counter and can define a carry out (fortunately you can use state 6 and do not require extra logic). You can use the carry out as an enable input to form two cascaded counters.

The carry out of 1D is used as an enable input to DICE2. Hence, 1D performs the same function as the push button switch performed in dice 1. Therefore, whenever 1D is enabled or logic high, DICE2 is enabled and rolls a number. DICE2 displays the number when 1D is disabled or logic is low. This configuration is shown in Figure 9.

```
                    Intel Programmable Logic System

Logic Builder Main Menu
F1 Help                         1a
F2 Exit                         1b   EQN   in1d                    INP NOJF
F3 New                          1c                                 EQN NORF
F4 Open                         1d                                CLKB NOSF
F5 Find                                                            AND NOTF
F6 Edit                                                           NAND ROIF
F7 Config                                                          NOR RONF
F8 Node List                                                       NOT RORF
F9 Redraw                                                           OR SONF
                                                                   XOR SOSF
                                                                  COIF TOIF
                                                                  CONF TONF
                                                                  JOJF TOTF
                                                                  JONF

-->

in1d=(1a*1b*1c*/1d);
in1d=(/1a*1b*1c*1d)*/switch+(/1a*1b*1c*1d)*switch;
```

Screen 11.

Table 8.

PROMPTS	USER ENTRY
Find: (Now use the <cursor left> key to obtain the EQN Primitive.) Edit: 1N1D = (/1A*1B*1C*1D)*/SWITCH+(1A*1B*1C*/1D)*SWITCH;<Enter>	1N1D <Enter>
Find: (Now use the <cursor left> key to obtain the EQN Primitive.) Edit: 1N1C = ((/1A*1B*1C*/1D)+(1A*1B*1C*/1D)+(/1A*1B*1C*1D))*/SWITCH +((1A*1B*/1C*/1D)+(/1A*1B*1C*/1D)+(1A*1B*1C*/1D))*SWITCH; <Enter>	1N1C <Enter>
Find: (Now use the <cursor left> key to obtain the EQN Primitive.) Edit: 1N1B = ((/1A*1B*/1C*/1D)+(1A*1B*/1C*/1D)+(/1A*1B*1C*1D)+(1A*1B*1C*/1D) +(/1A*1B*1C*1D))*/SWITCH +((1A*/1B*/1C*/1D)+(/1A*1B*/1C*/1D)+(1A*1B*/1C*/1D)+ (/1A*1B*1C*/1D)+(1A*1B*1C*/1D))*SWITCH;<Enter>	1N1B <Enter>
Find: (Now use the <cursor left> key to obtain the EQN Primitive.) Edit: 1N1A=((1A*/1B*/1C*/1D)+(1A*1B*/1C*/1D)+(1A*1B*1C*/1D)+ (/1A*/1B*/1C*/1D))*/SWITCH+((/1A*1B*/1C*/1D)+(/1A*1B*1C*/1D) +(/1A*1B*1C*1D)+(/1A*/1B*/1C*/1D))*SWITCH;<Enter>	1N1A <Enter>

The two conditions obtained are as follows:

When power is ON and 1D is enabled, DICE2 will roll.

When power is ON and 1D is disabled, DICE2 will display.

For DICE1, the logic conditions remain the same as in part A. Just as you used the switch to enable and disable

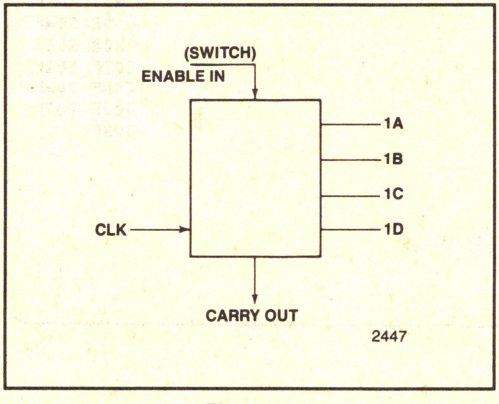


Figure 8.

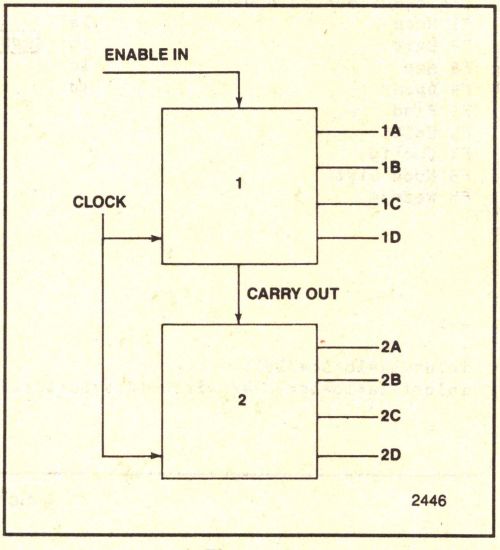


Figure 9.

DICE1, you will use the switch as well as the output of LED 1D to enable and disable DICE2; because the number on DICE2 is a function of both the switch and the present state of LED 1D, as explained above.

Now write down the truth table since the state diagrams can easily be inferred from the truth table. Please note that the truth table is identical to the one for DICE1 except for the switch input. For DICE2, you will have the combination of the switch and the 1D, as shown in Table 9.

The Boolean expressions for part C will consider the situation when the switch is ON as well as OFF and also 1D enabled or disabled respectively. The Boolean equations will contain the expression for the switch and LED 1D, as shown below.

```
DICE2A = ((2A*2B*/2C*/2D)+(2A*2B*/2C*/2D)
         +(2A*2B*2C*/2D)+(/2A*/2B*/2C*2D))
         *(/SWITCH*/1D)
         +((/2A*2B*/2C*/2D)+(/2A*2B*2C*/2D)
         +(/2A*2B*2C*2D)+(/2A*/2B*/2C*/2D))
         *(SWITCH*1D)

DICE2B = ((/2A*2B*/2C*/2D)+(2A*2B*/2C*/2D)
         +(/2A*2B*2C*/2D)+(2A*2B*2C*/2D)
         +(/2A*2B*2C*2D))*(/SWITCH*/1D)
         +((2A*/2B*/2C*/2D)
         +(/2A*2B*/2C*/2D)+(2A*2B*/2C*/2D)
         +(/2A*2B*2C*/2D)
         +(2A*2B*2C*/2D))*(SWITCH*1D)

DICE2C = ((/2A*2B*2C*/2D)+(2A*2B*2C*/2D)
         +(/2A*2B*2C*2D))*(/SWITCH*/1D)
         +((2A*2B*/2C*/2D)
         +(/2A*2B*2C*/2D)+(2A*2B*2C*/2D))
         *(SWITCH*1D)

DICE2D = (/2A*2B*2C*2D)*(/SWITCH*/1D)
         +(2A*2B*2C*/2D)*(SWITCH*1D)
```

Now you can use the iPLDS to program and test the device as explained in appendix C. At this stage in design, you have completed part C of the design which is to add a second DICE to the first one giving the the roll/no-roll option.

In part C of the design process, you have used one dedicated input which is the switch, and a total of eight output pins for the two pairs of LEDs, 1A, 1B, 1C, 1D and 2A, 2B, 2C, 2D respectively. You have also used the RORF primitive, since the design logic was the same for DICE2 as it was for DICE1. This leaves 3 dedicated inputs and 8 I/O pins on the 5C060 device.

You can stop the design now or go onto part D which gives the next option, which is adding the spin.

PART D

This is the fourth step in our design process and adds the spin option to the two dice that are rolling when the switch is pushed and display a number when the switch is released. The logic used to implement the spin concept is as follows:

When the power is ON and the switch is OFF, DICE1 and DICE2 display a random number according to the logic defined in parts B and C respectively.

But, when power is ON and the switch is ON, the two dice spin by lighting the LEDs B, C, and D. That is, DICE1 will light LEDs 1B, 1C, 1D while DICE2 will light LEDs 2B, 2C, and 2D. This pattern on the LEDs will generate the spinning pattern. The logic is shown in the truth table in Table 10. The schematic is shown in Figure 10.

As you can see from the truth table, when the present state is any of the three valid states, then the two dice will spin. The dice will also spin if the present state is an invalid state, because all the invalid states go to "0 0 0 0" in the next state. But from the truth table in Table 10, you see that this particular state is a valid state lighting LED C.

The spin frequency should be chosen to be visually appealing and should be high enough to ensure randomness of the dice. If we use the "carry out" state of DICE2, then the spin pattern will only change once for every combination of the two dice. This will ensure randomness. The "carry out" of DICE2 is signal 2d; we do not need extra terms to derive it.

Thus we have achieved our objective of adding the spinning option to the two dice.

The Boolean equations that are obtained from the above truth table are as follows:

```
SPIN1B = (SWITCH*2d*/S1D*/S1C*/S1B*S1A)

SPIN1C = (SWITCH*2d*/S1D*/S1C*/S1B*/S1A)

SPIN1D = (SWITCH*2d*/S1D*S1C*/S1B*/S1A)

SPIN2B = (SWITCH*2d*/S2D*/S2C*/S2B*S2A)

SPIN2C = (SWITCH*2d*/S2D*/S2C*/S2B*/S2A)

SPIN2D = (SWITCH*2d*/S2D*S2C*/S2B*/S2A)
```

Please note in the above equations that A, B, C, and D refer to both DICE1 and DICE2. For DICE1 the above set of equations would be 1A, 1B, 1C, and 1D. For DICE2 the above set of equations would be 2A, 2B, 2C, and 2D respectively. SD is the feedback obtained from IN D of both DICE1 and DICE2 respectively. If the switch is not ON, the dice will not spin and a random pair of numbers will be displayed by the two dice; but, if the switch is ON, then the two dice will spin according to the truth table and Boolean expression given in Table 10.

Table 9. Truth Table for DICE2

Input State					Output State							
(SWITCH*1D)	2A	2B	1C	2D	2A	2B	2C	2D	2A	2B	2C	2D
					Valid state				Invalid state			
REMAIN IN THE SAME STATE												
0	1	0	0	0	1	0	0	0(1)				
0	0	1	0	0	0	1	0	0(2)				
0	1	1	0	0	1	1	0	0(3)				
0	0	1	1	0	0	1	1	0(4)				
0	1	1	1	0	1	1	1	0(5)				
0	0	1	1	1	0	1	1	1(6)				
CONTROL THE INVALID STATES												
0	0	0	1	0					0	0	0	0
0	1	0	1	0					0	0	0	0
0	0	0	0	1					0	0	0	0
0	1	0	0	1					0	0	0	0
0	0	1	0	1					0	0	0	0
0	1	1	0	1					0	0	0	0
0	0	0	1	1					0	0	0	0
0	1	1	1	1					0	0	0	0
0	0	0	0	0					1	0	0	0(1)
CHANGE TO THE NEXT VALID STATE*												
1	1	0	0	0(1)	0	1	0	0(2)				
1	0	1	0	0(2)	1	1	0	0(3)				
1	1	1	0	0(3)	0	1	1	0(4)				
1	0	1	1	0(4)	1	1	1	0(5)				
1	1	1	1	0(5)	0	1	1	1(6)				
1	0	1	1	1(6)	1	0	0	0(1)				
CONTROL THE INVALID STATES												
1	0	0	1	0					0	0	0	0
1	1	0	1	0					0	0	0	0
1	0	0	0	1					0	0	0	0
1	1	0	0	1					0	0	0	0
1	0	1	0	1					0	0	0	0
1	1	1	0	1					0	0	0	0
1	0	0	1	1					0	0	0	0
1	1	0	1	1					0	0	0	0
1	1	1	1	1					0	0	0	0
1	0	0	0	0					1	0	0	0(1)

Note the extreme similarity between this truth table and the one given in Table 3.

Table 10. Truth Table to Spin Two Dice

Input State					Output State			
SWITCH	A	B	C	D	A	B	C	D
CHANGE TO THE NEXT VALID STATE								
1	0	0	0	0	0	0	1	0
1	0	0	0	1	0	1	0	0
1	0	0	1	0	0	0	0	1
1	0	1	0	0	0	0	0	0
ROLLING INTO A VALID STATE								
1	1	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0
1	0	1	1	0	0	0	0	0
1	1	1	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0
1	0	1	0	1	0	0	0	0
1	1	1	0	1	0	0	0	0
1	0	0	1	1	0	0	0	0
1	1	0	1	1	0	0	0	0
1	0	1	1	1	0	0	0	0
1	1	1	1	1	0	0	0	0

We have chosen the following two primitives for part D:

Registered Output Registered Feedback (RORF)

No output JK Feedback (NOJF)

For the dice spinning option you will use the RORF and for the dice not spinning option you will use the NOJF, while using the Logic Builder.

When you add the spinning option to the pair of rolling dice, you obtain the following boolean equations. (These Boolean equations satisfy the requirements of the two dice spinning when the switch is on and displaying a number when the switch is off).

```
SPIN1A = (/SWITCH*1A)

SPIN1B = (/SWITCH*1B)
         + (SWITCH*2d*/S1D*/S1C*/S1B*S1A)

SPIN1C = (/SWITCH*1C)
         + (SWITCH*2d*/S1D*/S1C*/S1B*/S1A)
         + (SWITCH*/2d*S1C)

SPIN1D = (/SWITCH*1D)
         +(SWITCH*2d*/S1D*S1C*/S1B*/S1A)

SPIN2A = (/SWITCH*2A)

SPIN2B = (/SWITCH*2B)
         + (SWITCH*2d*/S2D*/S2C*/S2B*S2A)

SPIN2C = (/SWITCH*2C)
         + (SWITCH*2d*/S2D*/S2C*/S2B*/S2A)

SPIN2D = (/SWITCH*2D)
         +(SWITCH*2d*/S2D*S2C*/S2B*/S2A)
```

At the end of the design step, you have completed all the design steps. You can now program the device using iPLDS.

The correct ADF file is included in Appendix D for your reference. You can refer to it to verify the ADF file you have created.

The programmed device can be tested on:

- A PCB with slow clock

For information on this board and on testing your design, please refer to Appendix C.

It works!

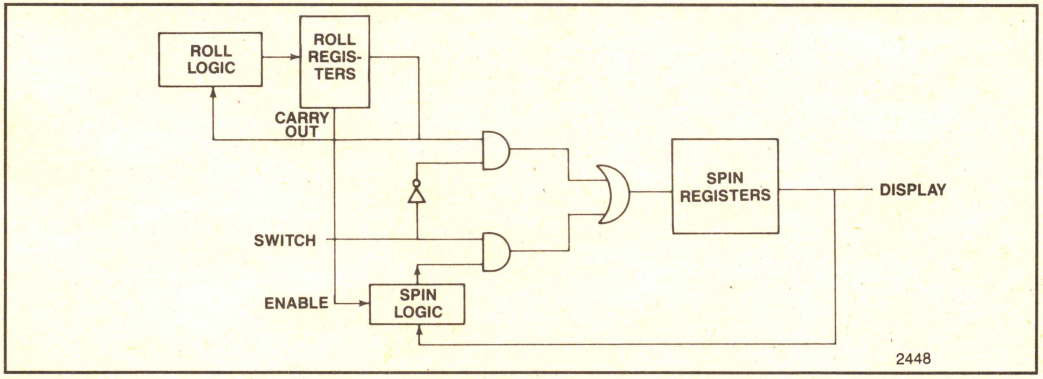


Figure 10.

LATE NEWS FLASH

The PCBs have been made and we have units in the field. Now Marketing wants the design updated! Field trials of the dice showed that the battery needed to last longer. A simple mod to the design, chop the drive to the LEDs, extends the battery life.

This is very simple using the EPLDs. Reprogram the EPLD and test it. Imagine how difficult it would have been without using EPLDs.

PART E: MORE POWER SAVINGS

This step of the design process is to modify the existing circuit to add the power save feature which will extend the battery life. This can easily be done by chopping the drive to the LEDs. Chopping the drive to the LEDs can be done as follows:

When you designed the circuit and implemented it using the iPLS, you have set the output enable (Oe) to VCC supply. This means that the LEDs are enabled 100% of the time. You can "chop" the drive to the LEDs with a conveniant high (above 50Hz) signal that will not be visible to the human eye.

To reduce current consumption by the LEDs, modify the NETWORK: Section of the ADF as follows:

1) Change the node name for the RORF Output Enables (Oe) from VCC to OE.
2) Insert an OENB (Output Enable Buffer) primitive to allow the logic array to drive the output enables.
3) Include an INP primitive to connect the output enable to an input pin.

These changes can appear as follows:

spin1a, s1a = RORF (in 1a, clock 1, GND, GND, OE)

OE = OENB (OEN)

OEN = INP (OEN)

Remember to include OEN in your INPUTS: declaration. The OEN input on the programmed part can now be connected to an appropriate clock signal to obtain the desired power savings.

CONCLUSION

You should now have a comprehensive knowledge of Intel's EPLD and iPLDS family of devices.

With this knowledge you will be able to implement designs using the iPLDS tools.

Good Luck!

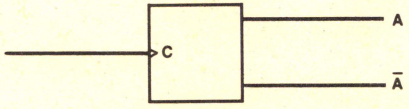
APPENDIX A:
BASIC DEFINITIONS

BASIC DEFINITIONS

Logic Design — A systematic procedure for realizing specified terminal characterisitics of digital networks, at either the device or system level.

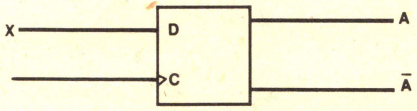
CLOCKED FLIP-FLOP — Output determined by the leading or trailing edge of clock pulse.

T FLIP-FLOP — Output changes value with every input clock pulse.



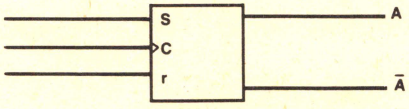
T FLIP-FLOP

D FLIP-FLOP — Output determined by the input signal when clock pulse present.



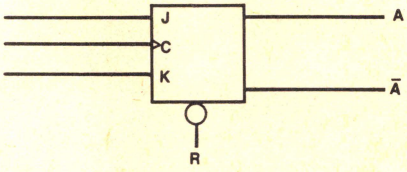
D FLIP-FLOP

S-R FLIP-FLOP — Output states synchronized with the clock pulse and controlled by the input signals, S and R.



S-R FLIP-FLOP

J-K FLIP-FLOP — Output states synchronized with the clock pulse and controlled by the input signals, J and K.



J-K FLIP-FLOP

COMBINATORIAL CIRCUIT — Output determined by current value of input signal.

REGISTERED CIRCUIT — Output determined by sequence of input signals.

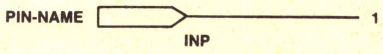
Intel Schematic Primitive — One of the basic functional blocks needed to design circuits for Intel programmable logic products.

Truth Table — A list of all the input-output possibilities of a logic circuit.

Boolean Logic — Describes logic that obeys the theorems of Boolean algebra. The Boolean portion of a design is that portion which can be implemented in the AND-OR matrix.

State Diagram — A diagram that shows the succession of output states through which the circuit passes as its input signals vary.

INP — Input

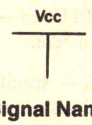


Input Primitive

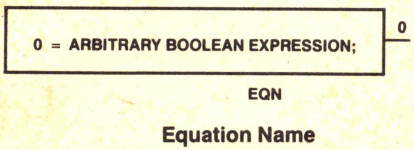
GND — Ground

Ground Signal Name

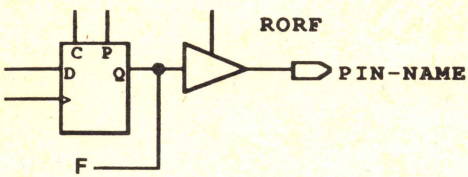
VCC — Signal



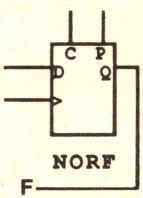
EQN — Equation



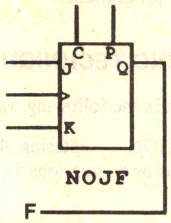
Registered Output Registered Feedback (RORF)



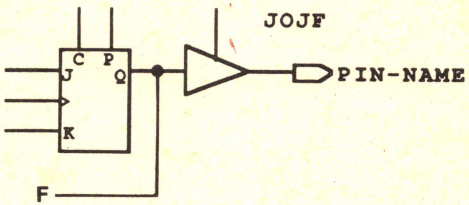
No Output Registered Feedback (NORF)



No Output JK Feedback (NOJF)



JK Output JK Feedback (JOJF)



Security Bit — A feature that prevents the device from being interrogated or being accidentally programmed.

Turbo-bit — A control bit that allows you to choose the speed and power characteristics of the device. If the inputs are static for approximately 50 ns and the Turbo-bit is not programmed, the device will enter power down mode. When the input changes, the device will take an extra 3-5 ns to wake-up and react to the change. Programming the Turbo-bit inhibits the power down.

Macrocell — A basic building block of Intel's programmable logic devices. A macrocell consists of two sections: combinatorial logic and output logic. The combinatorial logic allows a wide variety of logic functions. The output logic has two data paths: one leads to the other macrocells or feeds back to the macrocell itself: the other is configured as a pin configuration acting as input, output, or bi-directional I/O port on the chip.

Node — A wire connecting two or more primitives in a schematic.

Pin — A node that is connected to an input or I/O primitive on one end and a pin of the chip on the other end.

Product tem (P-Term) — Two or more factors in a boolean expression combined with the AND operator consitutes a logic product term.

JEDEC Standard File — An industry-wide standard for the transfer of information between a data preparation system and a logic device programmer.

EPLD PROGRAMMING TECHNIQUES

You can enter your design in the following ways

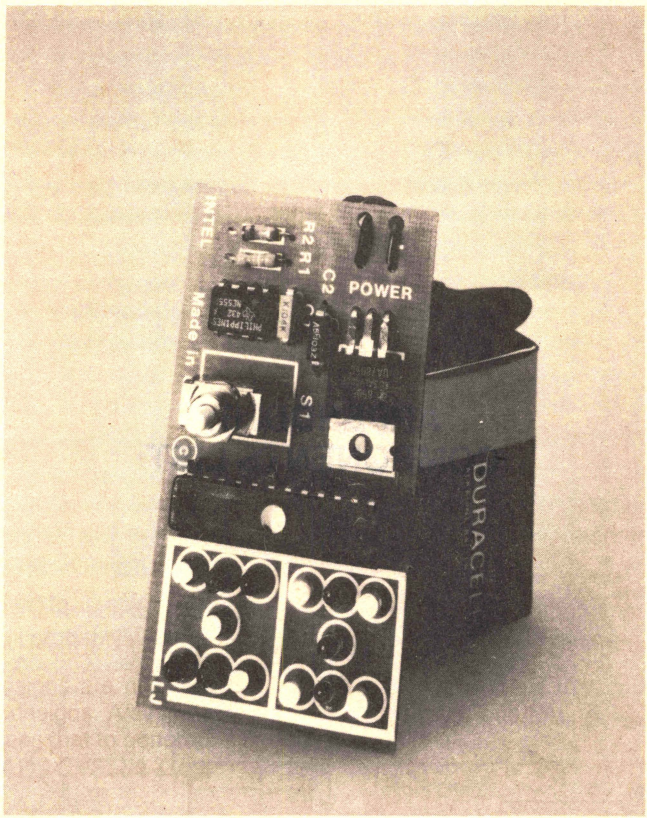
1. BOOLEAN EQUATION — entering the design in BOOLEAN equations or expressions.
2. NETLIST CAPTURE — selecting components and specifying interconnections until all elements are specified.
3. SCHEMATIC CAPTURE — using a mouse and menu driven environment.
4. STATE MACHINE — specifying states and conditional branches and also inputs/outputs to the state machines.

APPENDIX B:
COMPONENTS LIST

COMPONENTS USED IN DESIGN

In order to implement the EPLD program, you should use the following:

- An 5C060 EPLD
- A pair of seven discrete LEDs (Dice 1, Dice 2)
- A timer to generate a clock signal (NE555)
- A voltage regulator to generate a fixed voltage of 5 volts (7805)
- A push button switch to control the spinning mechanism
- A 9-Volt DC battery source to generate the power supply
- Capacitors C1 = 0.1 MF, C2 = 0.01 MF
- Resistors R1 = 390K, R2 = 100K
- A PCB as explained in Appendix C



APPENDIX C:
PCB DESCRIPTION

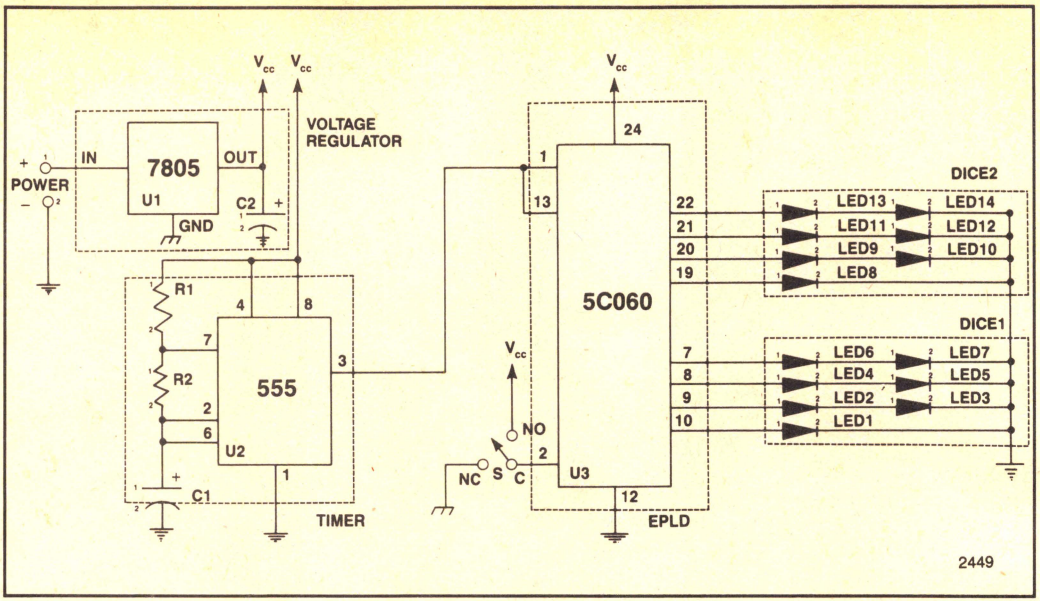


Figure C-1

You can test each part of your design using the PCB with a slow clock on it.

The PCB is a board that is very specific to the dice example. The PCB is portable, approximately 2″ × 3″. All the components except for the EPLD are easily available commercially. A complete list of all the components that are required for the PCB is given in Appendix B. The circuit can easily be connected and tested using the circuit diagram given below. After the four steps of the design are completed, the PCB can be used to throw a pair of dice in any home games such as Monopoly etc.

After the EPLD is programmed using the Logic Programmer, it can be inserted into the PCB. For design steps B, C, and D the push button switch can be used to generate the roll/no-roll or the spin/no spin option.

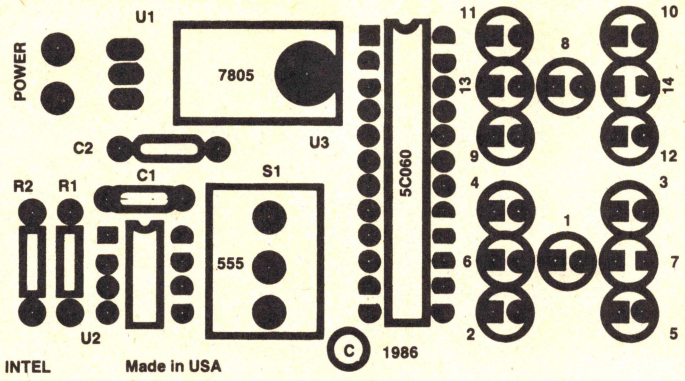
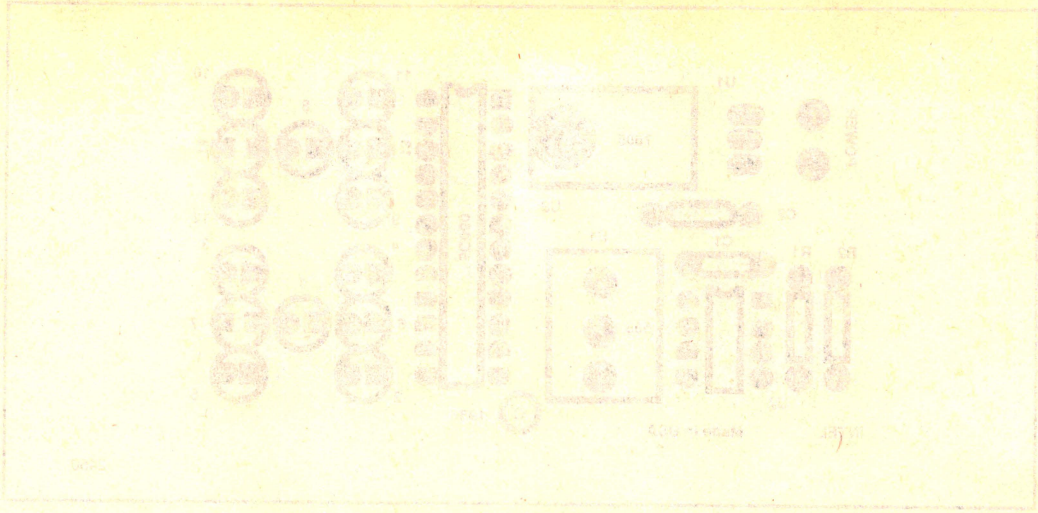

U1
POWER
7805
U3
C2
C1
S1
R2
R1
555
U2
5C060
11
10
8
13
14
9
12
4
3
1
6
7
2
5
INTEL
Made in USA
C
1986
2450

Figure C-2



APPENDIX D

ADF FOR PART A: SINGLE DICE ROLLING

```
Lakshmi Jayanthi
DSO Applications
February 19, 1986

5C060

Part A:   DICE ROLLING

LB Version 3.0, Baseline 17x, 9/26/85
PART: 5C060

INPUTS: clock1

OUTPUTS: dice1a@10,dice1b@9,dice1c@8,dice1d@7

NETWORK:

dice1a,1a = RORF (in1a,clock1,GND,GND,VCC)
dice1b,1b = RORF (in1b,clock1,GND,GND,VCC)
dice1c,1c = RORF (in1c,clock1,GND,GND,VCC)
dice1d,1d = RORF (in1d,clock1,GND,GND,VCC)

clock1 = INP (clock1)

EQUATIONS:

in1a =(/1a*1b*/1c*/1d)
        +(/1a*1b*1c*/1d)
        +(/1a*1b*1c*1d)
        +(/1a*/1b*/1c*/1d);
in1b =(1a*/1b*/1c*/1d)
        +(/1a*1b*/1c*/1d)
        +(1a*1b*/1c*/1d)
        +(/1a*1b*1c*/1d)
        +(1a*1b*1c*/1d);
in1c =(1a*1b*/1c*/1d)
        +(/1a*1b*1c*/1d)
        +(1a*1b*1c*/1d);
in1d =(1a*1b*1c*/1d);

END$
```

RPT FOR PART A: SINGLE DICE ROLLING

Logic Optimizing Compiler Utilization Report

***** Design implemented successfully

Lakshmi Jayanthi
DSO Applications
February 19, 1986

5C060

Part A: DICE ROLLING

LB Version 3.0, Baseline 17x, 9/26/85

```
                5C060
              - - - - -
    clock1 -| 1    24|- Vcc
       GND -| 2    23|- GND
       GND -| 3    22|- GND
       GND -| 4    21|- GND
       GND -| 5    20|- GND
       GND -| 6    19|- GND
    dice1d -| 7    18|- GND
    dice1c -| 8    17|- GND
    dice1b -| 9    16|- GND
    dice1a -|10    15|- GND
       GND -|11    14|- GND
       GND -|12    13|- GND
              - - - - -
```

INPUTS

Name	Pin	Resource	MCell #	PTerms	Feeds: MCells	OE	Clear	Clock
clock1	1	INP	-	-	-	-	-	CLK1

OUTPUTS

Name	Pin	Resource	MCell #	PTerms	Feeds: MCells	OE	Clear	Clock
dice1d	7	RORF	13	1/ 8	13 14 15 16	-	-	-
dice1c	8	RORF	14	2/ 8	13 14 15 16	-	-	-
dice1b	9	RORF	15	2/ 8	13 14 15 16	-	-	-
dice1a	10	RORF	16	2/ 8	13 14 15 16	-	-	-

```
**UNUSED RESOURCES**

      Name   Pin   Resource      MCell     PTerms

         -     2          -          -          -
         -     3          -          9          8
         -     4          -         10          8
         -     5          -         11          8
         -     6          -         12          8
         -    11          -          -          -
         -    13          -          -          -
         -    14          -          -          -
         -    15          -          8          8
         -    16          -          7          8
         -    17          -          6          8
         -    18          -          5          8
         -    19          -          4          8
         -    20          -          3          8
         -    21          -          2          8
         -    22          -          1          8
         -    23          -          -          -

**PART UTILIZATION**

22%      Pins
25%      MacroCells
5%       Pterms
```

NOTE: Since part A is a simple design, the part utilization is very low.

```
ADF FOR PART B: SINGLE DICE ROLL/NOT ROLL
-----------------------------------------

Lakshmi Jayanthi
DSO Applications
February 19, 1986


5C060

PART B: DICE ROLL AND NOT ROLL

LB Version 3.0, Baseline 17x, 9/26/85
PART: 5C060

INPUTS: clock1,switch@2

OUTPUTS: dice1a@10,dice1b@9,dice1c@8,dice1d@7

NETWORK:

dice1a,1a = RORF (in1a,clock1,GND,GND,VCC)
dice1b,1b = RORF (in1b,clock1,GND,GND,VCC)
dice1c,1c = RORF (in1c,clock1,GND,GND,VCC)
dice1d,1d = RORF (in1d,clock1,GND,GND,VCC)

clock1 = INP (clock1)

switch = INP(switch)

EQUATIONS:

in1a =(/1a*/1b*/1c*/1d*/switch)
        +(1a*/1b*/1c*/1d*/switch)
        +(1a*1b*/1c*/1d*/switch)
        +(1a*1b*1c*/1d*/switch)
        +(/1a*/1b*/1c*/1d*switch)
        +(/1a*1b*/1c*/1d*switch)
        +(/1a*1b*1c*/1d*switch)
        +(/1a*1b*1c*1d*switch);
in1b =(/1a*1b*/1c*/1d*/switch)
        +(1a*1b*/1c*/1d*/switch)
        +(/1a*1b*1c*/1d*/switch)
        +(1a*1b*1c*/1d*/switch)
        +(/1a*1b*1c*1d*/switch)
        +(1a*/1b*/1c*/1d*switch)
        +(/1a*1b*/1c*/1d*switch)
        +(1a*1b*/1c*/1d*switch)
        +(/1a*1b*1c*/1d*switch)
        +(1a*1b*1c*/1d*switch);
in1c =(/1a*1b*1c*/1d*/switch)
        +(1a*1b*1c*/1d*/switch)
        +(/1a*1b*1c*1d*/switch)
        +(1a*1b*/1c*/1d*switch)
        +(/1a*1b*1c*/1d*switch)
        +(1a*1b*1c*/1d*switch);
in1d =(/1a*1b*1c*1d*/switch)
        +(1a*1b*1c*/1d*switch);

END$
```

RPT FOR PART B: SINGLE DICE ROLL/NOT ROLL

Logic Optimizing Compiler Utilization Report

***** Design implemented successfully

Lakshmi Jayanthi
DSO Applications
February 19, 1986

5C060

PART B: DICE ROLL AND NOT ROLL

LB Version 3.0, Baseline 17x, 9/26/85

```
          5C060
         _ _ _ _ _
clock1 -| 1   24|- Vcc
switch -| 2   23|- GND
   GND -| 3   22|- GND
   GND -| 4   21|- GND
   GND -| 5   20|- GND
   GND -| 6   19|- GND
dice1d -| 7   18|- GND
dice1c -| 8   17|- GND
dice1b -| 9   16|- GND
dice1a -|10   15|- GND
   GND -|11   14|- GND
   GND -|12   13|- GND
         - - - - -
```

INPUTS

Name	Pin	Resource	MCell #	PTerms	Feeds: MCells	OE	Clear	Clock
clock1	1	INP	-	-	-	-	-	CLK1
switch	2	INP	-	-	13 14 15 16	-	-	-

OUTPUTS

Name	Pin	Resource	MCell #	PTerms	Feeds: MCells	OE	Clear	Clock
dice1d	7	RORF	13	2/ 8	13 14 15 16	-	-	-
dice1c	8	RORF	14	3/ 8	13 14 15 16	-	-	-

dice1b	9	RORF	15	3/ 8	13 14 15 16	-	-
dice1a	10	RORF	16	5/ 8	13 14 15 16	-	-

UNUSED RESOURCES

Name	Pin	Resource	MCell	PTerms
-	3	-	9	8
-	4	-	10	8
-	5	-	11	8
-	6	-	12	8
-	11	-	-	-
-	13	-	-	-
-	14	-	-	-
-	15	-	8	8
-	16	-	7	8
-	17	-	6	8
-	18	-	5	8
-	19	-	4	8
-	20	-	3	8
-	21	-	2	8
-	22	-	1	8
-	23	-	-	-

PART UTILIZATION

27% Pins
25% MacroCells
10% Pterms

NOTE: Part B of the design gets more complicated, hence the part utilization of the pins, macrocells and the Pterms is higher.

```
ADF FOR PART C: TWO DICE ROLLING
--------------------------------

Lakshmi Jayanthi
DSO Applications
February 19, 1986

5C060

PART C:  TWO DICE ROLL AND NOT ROLL

B Version 3.0, Baseline 17x, 9/26/85
PART: 5C060

INPUTS: clock1,clock2,switch@2

OUTPUTS: dice1a@10,dice1b@9,dice1c@8,dice1d@7,dice2a@19,dice2b@20,dice2c@21,dice
2d@22

NETWORK:

dice1a,1a = RORF (in1a,clock1,GND,GND,VCC)
dice1b,1b = RORF (in1b,clock1,GND,GND,VCC)
dice1c,1c = RORF (in1c,clock1,GND,GND,VCC)
dice1d,1d = RORF (in1d,clock1,GND,GND,VCC)

dice2a,2a = RORF (in2a,clock2,GND,GND,VCC)
dice2b,2b = RORF (in2b,clock2,GND,GND,VCC)
dice2c,2c = RORF (in2c,clock2,GND,GND,VCC)
dice2d,2d = RORF (in2d,clock2,GND,GND,VCC)

clock1 = INP (clock1)
clock2 = INP (clock2)

switch = INP (switch)

EQUATIONS:

in1a =(/1a*/1b*/1c*/1d*/switch)
        +(1a*/1b*/1c*/1d*/switch)
        +(1a*1b*/1c*/1d*/switch)
        +(1a*1b*1c*/1d*/switch)
        +(/1a*/1b*/1c*/1d*switch)
        +(/1a*1b*/1c*/1d*switch)
        +(/1a*1b*1c*/1d*switch)
        +(/1a*1b*1c*1d*switch);


in1b =(/1a*1b*/1c*/1d*/switch)
        +(1a*1b*/1c*/1d*/switch)
        +(/1a*1b*1c*/1d*/switch)
        +(1a*1b*1c*/1d*/switch)
        +(/1a*1b*1c*1d*/switch)
        +(1a*/1b*/1c*/1d*switch)
        +(/1a*1b*/1c*/1d*switch)
        +(1a*1b*/1c*/1d*switch)
        +(/1a*1b*1c*/1d*switch)
        +(1a*1b*1c*/1d*switch);
in1c =(/1a*1b*1c*/1d*/switch)
        +(1a*1b*1c*/1d*/switch)
        +(/1a*1b*1c*1d*/switch)
        +(1a*1b*/1c*/1d*switch)
        +(/1a*1b*1c*/1d*switch)
        +(1a*1b*1c*/1d*switch);
in1d =(/1a*1b*1c*1d*/switch)
        +(1a*1b*1c*/1d*switch);
```

```
in3a =(/2a*/2b*/2c*/2d*/(1d*switch))
      +(2a*/2b*/2c*/2d*/(1d*switch))
      +(2a*2b*/2c*/2d*/(1d*switch))
      +(2a*2b*2c*/2d*/(1d*switch))
      +(/2a*/2b*/2c*/2d*(1d*switch))
      +(/2a*2b*/2c*/2d*(1d*switch))
      +(/2a*2b*2c*/2d*(1d*switch))
      +(/2a*2b*2c*2d*(1d*switch));
in2b =(/2a*2b*/2c*/2d*/(1d*switch))
      +(2a*2b*/2c*/2d*/(1d*switch))
      +(/2a*2b*2c*/2d*/(1d*switch))
      +(2a*2b*2c*/2d*/(1d*switch))
      +(/2a*2b*2c*2d*/(1d*switch))
      +(2a*/2b*/2c*/2d*(1d*switch))
      +(/2a*2b*/2c*/2d*(1d*switch))
      +(2a*2b*/2c*/2d*(1d*switch))
      +(/2a*2b*2c*/2d*(1d*switch))
      +(2a*2b*2c*/2d*(1d*switch));
in2c =(/2a*2b*2c*/2d*/(1d*switch))
      +(2a*2b*2c*/2d*/(1d*switch))
      +(/2a*2b*2c*2d*/(1d*switch))
      +(2a*2b*/2c*/2d*(1d*switch))
      +(/2a*2b*2c*/2d*(1d*switch))
      +(2a*2b*2c*/2d*(1d*switch));
in2d =(/2a*2b*2c*2d*/(1d*switch))
      +(2a*2b*2c*/2d*(1d*switch));

END$
```

```
RPT FOR PART C: TWO DICE ROLLING
--------------------------------

Logic Optimizing Compiler Utilization Report

 ***** Design implemented successfully

Lakshmi Jayanthi
DSO Applications
February 19, 1986

5C060

PART C:  TWO DICE ROLL AND NOT ROLL

B Version 3.0, Baseline 17x, 9/26/85

                5C060
              - - - - -
      clock1 -| 1   24|- Vcc
      switch -| 2   23|- GND
         GND -| 3   22|- dice2d
         GND -| 4   21|- dice2c
         GND -| 5   20|- dice2b
         GND -| 6   19|- dice2a
      dice1d -| 7   18|- GND
      dice1c -| 8   17|- GND
      dice1b -| 9   16|- GND
      dice1a -|10   15|- GND
         GND -|11   14|- GND
         GND -|12   13|- clock2
              - - - - -

**INPUTS**

                                                    |          Feeds:
      Name  Pin  Resource    MCell #     PTerms    |  MCells     OE  Clear  Clock

    clock1    1       INP          -          -    |        -      -      -   CLK1

    switch    2       INP          -          -    |        1      -      -      -
                                                    |        2
                                                    |        3
                                                    |        4
                                                    |       13
                                                    |       14
                                                    |       15
                                                    |       16

    clock2   13       INP          -          -    |        -      -      -   CLK2


**OUTPUTS**

                                                    |          Feeds:
      Name  Pin  Resource    MCell #     PTerms    |  MCells     OE  Clear  Clock

    dice1d    7      RORF         13       2/ 8    |        1      -      -      -
                                                    |        2
                                                    |        3
                                                    |        4
                                                    |       13
                                                    |       14
                                                    |       15
                                                    |       16
```

dice1c	8	RORF	14	3/ 8	13 14 15 16	-	-	-
dice1b	9	RORF	15	3/ 8	13 14 15 16	-	-	-
dice1a	10	RORF	16	5/ 8	13 14 15 16	-	-	-
dice2a	19	RORF	4	7/ 8	1 2 3 4	-	-	-
dice2b	20	RORF	3	4/ 8	1 2 3 4	-	-	-
dice2c	21	RORF	2	4/ 8	1 2 3 4	-	-	-
dice2d	22	RORF	1	3/ 8	1 2 3 4	-		

UNUSED RESOURCES

Name	Pin	Resource	MCell	PTerms
-	3	-	9	8
-	4	-	10	8
-	5	-	11	8
-	6	-	12	8
-	11	-	-	-
-	14	-	-	-
-	15	-	8	8
-	16	-	7	8
-	17	-	6	8
-	18	-	5	8
-	23	-	-	-

PART UTILIZATION

50% Pins
50% MacroCells
24% Pterms

NOTE: in part C of the design you have added the second dice. Hence you can see that fifty percent of the device has been used.

```
ADF FOR PART D: TWO DICE SPINNING
---------------------------------

Lakshmi Jayanthi
DSO Applications
February 19, 1986


5C060

PART D: TWO DICE SPINNING

B Version 3.0, Baseline 17x, 9/26/85
PART: 5C060

INPUTS: clock1,clock2,switch@2

OUTPUTS: spin1a@10,spin1b@9,spin1c@8,spin1d@7,spin2a@19,spin2b@20,spin2c@21,spir
2d@22

NETWORK:

1a = NOJF (in1a,clock1,in11a,GND,GND)
1b = NOJF (in1b,clock1,in11b,GND,GND)
1c = NOJF (in1c,clock1,in11c,GND,GND)
1d = NOJF (in1d,clock1,in11d,GND,GND)

2a = NOJF (in2a,clock2,in22a,GND,GND)
2b = NOJF (in2b,clock2,in22b,GND,GND)
2c = NOJF (in2c,clock2,in22c,GND,GND)
2d = NOJF (in2d,clock2,in22d,GND,GND)

in11a = NOT(in1a)
in11b = NOT(in1b)
in11c = NOT(in1c)
in11d = NOT(in1d)

in22a = NOT(in2a)
in22b = NOT(in2b)
in22c = NOT(in2c)
in22d = NOT(in2d)

spin1a,s1a = RORF (ins1a,clock1,GND,GND,VCC)
spin1b,s1b = RORF (ins1b,clock1,GND,GND,VCC)
spin1c,s1c = RORF (ins1c,clock1,GND,GND,VCC)
spin1d,s1d = RORF (ins1d,clock1,GND,GND,VCC)

spin2a,s2a = RORF (ins2a,clock2,GND,GND,VCC)
spin2b,s2b = RORF (ins2b,clock2,GND,GND,VCC)
spin2c,s2c = RORF (ins2c,clock2,GND,GND,VCC)
spin2d,s2d = RORF (ins2d,clock2,GND,GND,VCC)

clock1 = INP (clock1)
clock2 = INP (clock2)

switch = INP (switch)

EQUATIONS:

in1a =(/1a*/1b*/1c*/1d*/switch)
        +(1a*/1b*/1c*/1d*/switch)
        +(1a*1b*/1c*/1d*/switch)
        +(1a*1b*1c*/1d*/switch)
```

```
          +(/1a*/1b*/1c*/1d*switch)
          +(/1a*1b*/1c*/1d*switch)
          +(/1a*1b*1c*/1d*switch)
          +(/1a*1b*1c*1d*switch);
in1b =(/1a*1b*/1c*/1d*/switch)
          +(1a*1b*/1c*/1d*/switch)
          +(/1a*1b*1c*/1d*/switch)
          +(1a*1b*1c*/1d*/switch)
          +(/1a*1b*1c*1d*/switch)
          +(1a*/1b*/1c*/1d*switch)
          +(/1a*1b*/1c*/1d*switch)
          +(1a*1b*/1c*/1d*switch)
          +(/1a*1b*1c*/1d*switch)
          +(1a*1b*1c*/1d*switch);
in1c =(/1a*1b*1c*/1d*/switch)
          +(1a*1b*1c*/1d*/switch)
          +(/1a*1b*1c*1d*/switch)
          +(1a*1b*/1c*/1d*switch)
          +(/1a*1b*1c*/1d*switch)
          +(1a*1b*1c*/1d*switch);
in1d =(/1a*1b*1c*1d*/switch)
          +(1a*1b*1c*/1d*switch);

in2a =(/2a*/2b*/2c*/2d*/(1d*switch))
          +(2a*/2b*/2c*/2d*/(1d*switch))
          +(2a*2b*/2c*/2d*/(1d*switch))
          +(2a*2b*2c*/2d*/(1d*switch))
          +(/2a*/2b*/2c*/2d*(1d*switch))
          +(/2a*2b*/2c*/2d*(1d*switch))
          +(/2a*2b*2c*/2d*(1d*switch))
          +(/2a*2b*2c*2d*(1d*switch));
in2b =(/2a*2b*/2c*/2d*/(1d*switch))
          +(2a*2b*/2c*/2d*/(1d*switch))
          +(/2a*2b*2c*/2d*/(1d*switch))
          +(2a*2b*2c*/2d*/(1d*switch))
          +(/2a*2b*2c*2d*/(1d*switch))
          +(2a*/2b*/2c*/2d*(1d*switch))
          +(/2a*2b*/2c*/2d*(1d*switch))
          +(2a*2b*/2c*/2d*(1d*switch))
          +(/2a*2b*2c*/2d*(1d*switch))
          +(2a*2b*2c*/2d*(1d*switch));
in2c =(/2a*2b*2c*/2d*/(1d*switch))
          +(2a*2b*2c*/2d*/(1d*switch))
          +(/2a*2b*2c*2d*/(1d*switch))
          +(2a*2b*/2c*/2d*(1d*switch))
          +(/2a*2b*2c*/2d*(1d*switch))
          +(2a*2b*2c*/2d*(1d*switch));
in2d =(/2a*2b*2c*2d*/(1d*switch))
          +(2a*2b*2c*/2d*(1d*switch));

ins1a = (/switch*1a);
ins1b = (/switch*1b)
          +((2d*switch)*s1d*/s1c*/s1b*/s1a);
ins1c = (/switch*1c)
          +((2d*switch)*/s1a*/s1b*/s1c*/s1d);
ins1d = (/switch*1d)
          +((2d*switch)*/s1a*/s1b*s1c*/s1d);

ins2a = (/switch*2a);
ins2b = (/switch*2b)
          +((2d*switch)*s2d*/s2c*/s2b*/s2a);
ins2c = (/switch*2c)
          +((2d*switch)*/s2a*/s2b*/s2c*/s2d);
ins2d = (/switch*2d)
          +((2d*switch)*/s2a*/s2b*s2c*/s2d);

END$
```

```
LEF FOR PART D: TWO DICE SPINNING
---------------------------------

Lakshmi Jayanthi
DSO Applications
February 19, 1986


5C060

PART:
        5C060

INPUTS:

        clock1, clock2, switch@2


OUTPUTS:

        spin1a@10, spin1b@9, spin1c@8, spin1d@7, spin2a@19, spin2b@20,
        spin2c@21, spin2d@22

NETWORK:

        clock1 = INP(clock1)
        clock2 = INP(clock2)

        switch = INP(switch)

        spin1a, s1a = RORF(ins1a, clock1, GND, GND, VCC)
        spin1b, s1b = RORF(ins1b, clock1, GND, GND, VCC)
        spin1c, s1c = RORF(ins1c, clock1, GND, GND, VCC)
        spin1d, s1d = RORF(ins1d, clock1, GND, GND, VCC)

        spin2a, s2a = RORF(ins2a, clock2, GND, GND, VCC)
        spin2b, s2b = RORF(ins2b, clock2, GND, GND, VCC)
        spin2c, s2c = RORF(ins2c, clock2, GND, GND, VCC)
        spin2d, s2d = RORF(ins2d, clock2, GND, GND, VCC)

        %  ***  Resource, NOJF, was minimized to NORF  ***  %

        2d = NORF(..SG007D, clock2, GND, GND)

        %  ***  Resource, NOJF, was minimized to NOTF  ***  %

        2c = NOTF(..SG006D, clock2, GND, GND)

        %  ***  Resource, NOJF, was minimized to NORF  ***  %

        2b = NORF(..SG005D, clock2, GND, GND)

        %  ***  Resource, NOJF, was minimized to NORF  ***  %

        2a = NORF(..SG004D, clock2, GND, GND)

        %  ***  Resource, NOJF, was minimized to NORF  ***  %

        1d = NORF(..SG003D, clock1, GND, GND)

        %  ***  Resource, NOJF, was minimized to NORF  ***  %

        1c = NORF(..SG002D, clock1, GND, GND)
```

```
% *** Resource, NOJF, was minimized to NORF *** %

1b = NORF(..SG001D, clock1, GND, GND)

% *** Resource, NOJF, was minimized to NORF *** %

1a = NORF(..SG000D, clock1, GND, GND)

EQUATIONS:

ins2d = switch' * 2d
      + 2d * switch * s2a' * s2b' * s2c * s2d';

ins2c = switch' * 2c
      + 2d * switch * s2a' * s2b' * s2c' * s2d';

ins2b = switch' * 2b
      + 2d * switch * s2d * s2c' * s2b' * s2a';

ins2a = switch' * 2a

ins1d = switch' * 1d
      + 2d * switch * s1a' * s1b' * s1c * s1d';

ins1c = switch' * 1c
      + 2d * switch * s1a' * s1b' * s1c' * s1d';

ins1b = switch' * 1b
      + 2d * switch * s1d * s1c' * s1b' * s1a';

ins1a = switch' * 1a;

..SG000D = 1a' * 1b' * 1c' * 1d'
         + 1a * 1c' * 1d' * switch'
         + 1a' * 1b * 1d' * switch
         + 1a * 1b * 1d' * switch'
         + 1a' * 1b * 1c * switch;

..SG001D = 1b * 1d'
         + 1b * 1a' * 1c * switch'
         + 1a * 1c' * 1d' * switch;

..SG002D = 1c * 1b * 1d'
         + 1c * 1a' * 1b * switch'
         + 1a * 1b * 1d' * switch;

..SG003D = 1d * 1a' * 1b * 1c * switch'
         + 1d' * 1a * 1b * 1c * switch;

..SG004D = 2a' * 2b' * 2c' * 2d'
         + 2a * 2c' * 2d' * 1d'
         + 2a * 2c' * 2d' * switch'
         + 2a * 2b * 2d' * 1d'
         + 2a * 2b * 2d' * switch'
         + 2a' * 2b * 2d' * 1d * switch
         + 2a' * 2b * 2c * 1d * switch;

..SG005D = 2b * 2d'
         + 2b * 2a' * 2c * 1d'
         + 2b * 2a' * 2c * switch'
         + 2a * 2c' * 2d' * 1d * switch;
```

```
..SG006D = 2c * 2b'
         + 2c * 2a * 2d
         + 2c * 2d * 1d * switch
         + 2c' * 2a * 2b * 2d' * 1d * switch;

..SG007D = 2d * 2a' * 2b * 2c * switch'
         + 2d * 2a' * 2b * 2c * 1d'
         + 2d' * 2a * 2b * 2c * 1d * switch;

END$
```

NOTE: PLease note how the IPLS software has simplified the equations for you. You need not worry about minimization. The complicated Boolean expressions have been minimized to a great extent.

RPT FOR PART D: TWO DICE SPINNING

Logic Optimizing Compiler Utilization Report

***** Design implemented successfully

Lakshmi Jayanthi
DSO Applications
February 19, 1986

5C060

PART D: TWO DICE SPINNING

B Version 3.0, Baseline 17x, 9/26/85

```
                5C060
               _ _ _ _ _
     clock1 -| 1   24|- Vcc
     switch -| 2   23|- GND
   RESERVED -| 3   22|- spin2d
   RESERVED -| 4   21|- spin2c
   RESERVED -| 5   20|- spin2b
   RESERVED -| 6   19|- spin2a
     spin1d -| 7   18|- RESERVED
     spin1c -| 8   17|- RESERVED
     spin1b -| 9   16|- RESERVED
     spin1a -|10   15|- RESERVED
        GND -|11   14|- GND
        GND -|12   13|- clock2
               - - - - -
```

INPUTS

Name	Pin	Resource	MCell #	PTerms	Feeds: MCells	Feeds: OE	Feeds: Clear	Feeds: Clock
clock1	1	INP	-	-	-	-	-	CLK1
switch	2	INP	-	-	1	-	-	-
					2			
					3			
					4			
					5			
					6			
					7			
					8			
					9			
					10			
					11			
					12			
					13			
					14			
					15			
					16			
clock2	13	INP	-	-	-	-	-	CLK2

OUTPUTS

Name	Pin	Resource	MCell #	PTerms	Feeds: MCells	OE	Clear	Clock
spin1d	7	RORF	13	2/ 8	13 14 15	-	-	-
spin1c	8	RORF	14	2/ 8	13 14 15	-	-	-
spin1b	9	RORF	15	2/ 8	13 14 15	-	-	-
spin1a	10	RORF	16	1/ 8	13 14 15	-	-	-
spin2a	19	RORF	4	1/ 8	1 2 3	-	-	-
spin2b	20	RORF	3	2/ 8	1 2 3	-	-	-
spin2c	21	RORF	2	2/ 8	1 2 3	-	-	-
spin2d	22	RORF	1	2/ 8	1 2 3	-	-	-

BURIED REGISTERS

Name	Pin	Resource	MCell #	PTerms	Feeds: MCells	OE	Clear	Clock
	18	NORF	5	3/ 8	1 5 6 7 8	-	-	-
	17	NORF	6	4/ 8	2 5 6 7 8	-	-	-
	16	NORF	7	4/ 8	3 5 6 7 8	-	-	-

```
    15        NORF        8        7/ 8        4        -        -        -
                                                 5
                                                 6
                                                 7
                                                 8

     3        NORF        9        2/ 8        5        -        -        -
                                                 6
                                                 7
                                                 8
                                                 9
                                                10
                                                11
                                                12
                                                13

     4        NORF       10        3/ 8        9        -        -        -
                                                10
                                                11
                                                12
                                                14

     5        NORF       11        3/ 8        9        -        -        -
                                                10
                                                11
                                                12
                                                15

     6        NORF       12        5/ 8        9        -
                                                10
                                                11
                                                12
                                                16
```

```
**UNUSED RESOURCES**

      Name  Pin  Resource      MCell     PTerms

         -   11         -          -          -
         -   14         -          -          -
         -   23         -          -          -

**PART UTILIZATION**

86%      Pins
100%     MacroCells
35%      Pterms
```

NOTE: Part D of the design example utilizes the device in a very optimum manner. You have utilized all the macrocells and also 86% of the pins but only 35% of the product terms.

You have not used three of the input pins.

Consider this:

Make these three pins a mode select on this dice example — if all of these three additional inputs are high then the dice will function as described (this condition must be added to each product term). You now have seven other modes in which to operate this DICE. Anyone want to "load" the odds for "boxcars" or "snake-eyes"? You have 65% more product terms to use so you can be very creative. What else could you add to this EPLD?

APPENDIX E:
ADDITIONAL PUBLICATIONS

ADDITIONAL PUBLICATIONS

The following publications available from Intel contain additional information on Intel's EPLD family of parts and also on the iPLDS system.

- *5C031/032* data sheet, order number 290110-001
- *5C060/090* data sheet, order number 290104-002
- *5C060* errata sheet, order number 2906030-001
- *5C121* data sheet, order number 290098-002
- *iPLDS* data sheet, order number 280168-002
- *Intel Programmable Logic Software User Manual,* order number 166612

The PCB card and diskette with pre-entered design files may be obtained through Intel's Literature Distribution for a nominal fee. To order, request:

iPLDS DICE DEMO BOARD

Please write to:

Intel Literature Distribution
SC6-714
3065 Bowers Avenue
Santa Clara, CA 95051,

or call toll-free (800) 548-4725 for additional information.

intel

APPLICATION NOTE

AP-311

May 1987

Using Macros in EPLD Designs

DANIEL E. SMITH
APPLICATIONS ENGINEERING
INTEL CORPORATION

Order Number: 292039-001

INTRODUCTION

The iPLS II (Intel Programmable Logic Software) Logic Optimizing Compiler includes a Macro Expander that supports the use of macros in EPLD designs. This application note shows how to use the TTL and Gate-Array macros available from Intel with ADFs created by a text editor. Included are descriptions of macro file support, guidelines for using macros, and three design examples.

OVERVIEW

iPLS II allows designers to include macro calls in design files to implement common circuit functions. Macro calls are subsequently expanded by the LOC (Logic Optimizing Compiler) into ADF network and/or equation entries required to perform the desired functions. Use of macros allows designs to proceed at a high level, which simplifies and shortens the design process. Macros can be connected together or used in conjunction with standard iPLS II EPLD primitives. Designing with macros is analogous in many ways to using subroutines in software.

Macros can be used in ADFs (Advanced Design Files) created by a text editor, or by several schematic capture software products. This application note covers use of macros in ADFs created by a text editor. Macro support at this level includes the following:

- A TTL macro library (TTL.LIB) for designing with common TTL circuit equivalents
- A gate-array macro library (INTEL.LIB) for designing with common gate-array primitives.
- A Macro Expander in the LOC that expands macro calls in ADFs with the contents of the corresponding macros from libraries.

Figure 1 shows text editor/ADF macro support for iPLS II. Note that the ADF can be created by any standard ASCII text editor (text editor supplied by user). Creation of user-defined macros is covered in application note, AP-312 "Creating Macros for EPLD Designs", order number 292040. Use of macros with schematic capture software is covered in the documentation for the respective software package.

This note discusses use of macros under the following headings:

- Macro Libraries, briefly describes the two libraries available from Intel.
- Using Macros, describes macro files, how to call macros, the process of macro expansion, calling multiple macro calls, and some basic guidelines to follow and pitfalls to avoid.
- Three examples showing use of TTL macros, gate array macros, and mixing macros and EPLD primitives.

MACRO LIBRARIES

Intel offers two macro libraries: a TTL Library and a Gate Array Library.

TTL Macro Library

A TTL macro library (TTL.LIB) is available from Intel to support design entry using familiar 74-series logic

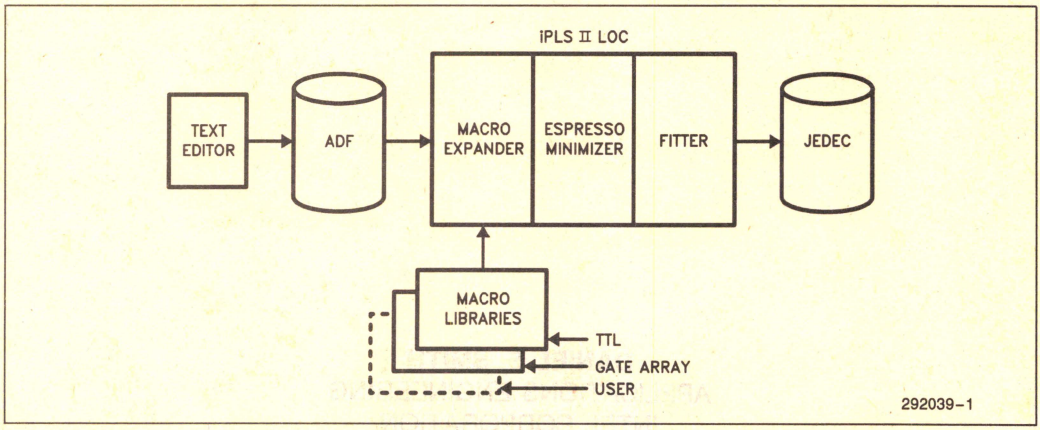


Figure 1. Text Editor/ADF Macro Support for iPLS II

devices. The library contains macros that implement the most widely used 74-series device functions as well as macros for some members of other logic families. Each device in the library is supported by a .DOC file. The .DOC file describes the macro syntax and lists any notable differences between the macro implementation and the TTL part.

Gate Array Macro Library

A Gate Array macro library (INTEL.LIB) is available from Intel to support design entry using familiar gate array design primitives. These macros implement almost all of the gate array macrocell functions in the Intel Gate Array Macrocell Library. The macros are supported by the "Intel Gate Array Library User Handbook" document file (filename GA_USRBK.LST).

USING MACROS

The iPLS II Macro Expander is automatically invoked by the LOC when an ADF is submitted to the compiler. When invoked, the Macro Expander identifies macro calls in ADFs, searches macro libraries for a corresponding macro, and expands the call with ADF network and equation entries from the macro file. The expanded file is then compiled normally.

Macro Files

Figure 2 shows the macro file for a 74138 TTL device, a commonly used one-of-eight decoder. Note that the first line contains the name and I/O signals for the device. Signals are listed in the order in which they appear on the actual TTL device, including VCC and GND (i.e., A = pin 1, B = pin 2, ..., VCC = pin 16). The sequence of signals in this line determines how the macro is "called" from an ADF.

Some of the macros in the TTL library have an "X" suffix appended to the filename, for example 74138X. This suffix indicates that the macro is device-specific (not supported on all EPLDs) or that there is some difference from the TTL device. This information is described in the .DOC file for each macro.

The second line of the macro file contains defaults for each input and place holders (blanks) for each output. The default for an input sets the input to an intelligent level (i.e., enables are enabled, clears, preset, loads are disabled, etc.).

Macro files can contain a Network section, an Equation section, or both. A Network section is not needed when the macro functions can all be implemented in Boolean equations. When used, the Network section contains EPLD design primitives. An Equations section is not needed when the macro functions can all be implemented in the Network section. Macro files end with the keyword "ENDEF".

Macro Calls

All macro calls appear in the Network section of an ADF. Macro calls use the same part/function name and signal sequence used on the first line of the macro file. The signal names in the macro and the macro call do not need to match, but the order of signals in the call is **crucial** to proper implementation of the macro function. For example, the macro call for the 74138 device could be any one of the following examples:

```
74138(A,B,C,G2A,G2B,G1,Y7,GND,Y6,Y5,
Y4,Y3,Y2,Y1,Y0,VCC)

74138(D1,D2,D3,EN1,EN2,EN3,07,GND,06,
05,04,03,02,01,00,VCC)
```

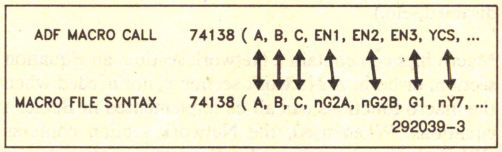
```
74138(A,B,C,nG2A,nG2B,G1,nY7,GND,nY6,nY5,nY4,nY3,nY2,nY1,nY0,VCC)
DEFAULT:(GND,GND,GND,GND,GND,VCC,,GND,,,,,,,,VCC)

NETWORK:

EQUATIONS:

nY0   = !(!A * !B * !C * !nG2A * !nG2B * G1);
nY1   = !(A * !B * !C * !nG2A * !nG2B * G1);
nY2   = !(!A * B * !C * !nG2A * !nG2B * G1);
nY3   = !(A * B * !C * !nG2A * !nG2B * G1);
nY4   = !(!A * !B * C * !nG2A * !nG2B * G1);
nY5   = !(A * !B * C * !nG2A * !nG2B * G1);
nY6   = !(!A * B * C * !nG2A * !nG2B * G1);
nY7   = !(A * B * C * !nG2A * !nG2B * G1);

ENDEF
$
```

292039-2

Figure 2. Sample TTL Macro File (74138.DEV)

```
74138(A,B,C,ENA,ENB,ENC,Y7,GND,Y6,Y5,
Y4,Y3,Y2,Y1,Y0,VCC)
```

In each case, the part name corresponds to the macro part name. The names of the signals differ, but the order of signals match the macro. During processing, the Macro expander assigns node connections between the macro call and the macro file based on the positions of signals, not the names of the signals. For example, note the following macro call to macro file signal assignments:



TTL macro signals originating outside the target EPLD require a prior INPUT macro call in the Network section. All signals used as outputs require a prior OUTPUT macro call in the Network section. Figure 3 shows a sample ADF that uses the 74138 macro. Each input is listed in the INPUTS: declaration and has an INPUT macro call. Outputs are listed in the OUTPUTS: declaration and have OUTPUT macro calls. (EPLD INP and CONF primitive statements may also be used in place of INPUT and OUTPUT macro calls, if desired.)

Gate arrays support a much richer selection of input and output types than EPLDs. Gate array signals originating outside the target gate array device require the appropriate gate array input or output macro calls. When using gate array macros with EPLDs, the I/O macros are implemented in terms of EPLD primitives. Note that when designs targeted for gate arrays are partitioned for multiple EPLDs, many internal gate array signals are transformed into EPLD input and output signals. These signals must be supported by INPUT and OUTPUT macro calls.

Macro Expansion

When the Macro Expander is invoked (by the LOC), it expands macro calls with the ADF network and/or equations entries from macro libraries (the TTL library in the case of the 74138). The Macro Expander searches libraries in the following order:

- user libraries (**filename**.LIB)
- TTL macro library (TTL.LIB)
- gate-array library (INTEL.LIB)

First, the Macro Expander searches in the current directory for MACRO.LIB, then along the "IPLS" environment variable for the user libraries specified there. Next, it searches for TTL.LIB in the IPLSII directory. Finally, it searches for INTEL.LIB in the IPLSII directory. The first occurrence of a macro is used.

```
YOUR NAME
YOUR COMPANY
DATE
1
A
5C060
One-of-Eight Decoder

OPTIONS: TURBO=OFF
PART: 5C060
INPUTS: A,B,C,G2A,G2B,G1
OUTPUTS: Y7,Y6,Y5,Y4,Y3,Y2,Y1,Y0

NETWORK:

INPUT(A,A)
INPUT(B,B)
INPUT(C,C)
INPUT(G2A,G2A)
INPUT(G2B,G2B)
INPUT(G1,G1)
OUTPUT(Y7,Y7)
OUTPUT(Y6,Y6)
OUTPUT(Y5,Y5)
OUTPUT(Y4,Y4)
OUTPUT(Y3,Y3)
OUTPUT(Y2,Y2)
OUTPUT(Y1,Y1)
OUTPUT(Y0,Y0)
74138(A,B,C,G2A,G2B,G1,Y7,GND,Y6,Y5,Y4,Y3,Y2,Y1,Y0,VCC)

END$
```

292039-4

Figure 3. ADF File Calling the 74138 Macro

The Macro Expander uses the ADF Network and Equation entries from the macro libraries and assigns the appropriate primitives for INPUT and OUTPUT calls. INP primitives are assigned to replace the INPUT macro calls. The OUTPUT calls are assigned primitives with output pins and output enables are supplied where needed.

Combination of primitives is automatically performed when needed. For example, when a feedback primitive such as a NORF feeds an output primitive such as a RONF, the Macro Expander combines the two primitives into a RORF. Combination of primitives conserves resources and results in the shortest possible delay path through the device.

During macro expansion, unused nodes are eliminated. For example, the VCC and GND nodes that correspond to TTL power and ground pins are eliminated. If an input node is not connected to a node in the ADF, the default value for that node is assigned from the DEFAULT: section of the macro file. Note, however, that the default value for each input in the macro file may be the value that disables the input or, for data inputs, is usually a logic 0. To be certain of the level used, specify a "VCC" or "GND" in the macro call for unused inputs.

The INPUT and OUTPUT calls and the original macro call are "commented out" by surrounding them with percent (%) signs. The %^% string is placed at the start of lines where primitives are created by the Macro Expander. The fully expanded file is written to the disk using the original filename and a .SDF extension. Figure 4 shows the Network and Equation sections for the 74138 SDF.

One final note with regard to compiling ADFs that use macros. Warning messages are typically encountered

```
NETWORK:

%    INPUT(A,A)    %
%^% A=INP(A)
%    INPUT(B,B)    %
%^% B=INP(B)
%    INPUT(C,C)    %
%^% C=INP(C)
%    INPUT(G2A,G2A)    %
%^% G2A=INP(G2A)
%    INPUT(G2B,G2B)        %
%^% G2B=INP(G2B)
%    INPUT(G1,G1)    %
%^% G1=INP(G1)
%    OUTPUT(Y7,Y7)    %
%^% Y7=CONF(Y7,VCC)
%    OUTPUT(Y6,Y6)    %
%^% Y6=CONF(Y6,VCC)
%    OUTPUT(Y5,Y5)    %
%^% Y5=CONF(Y5,VCC)
%    OUTPUT(Y4,Y4)    %
%^% Y4=CONF(Y4,VCC)
%    OUTPUT(Y3,Y3)    %
%^% Y3=CONF(Y3,VCC)
%    OUTPUT(Y2,Y2)    %
%^% Y2=CONF(Y2,VCC)
%    OUTPUT(Y1,Y1)    %
%^% Y1=CONF(Y1,VCC)
%    OUTPUT(Y0,Y0)    %
%^% Y0=CONF(Y0,VCC)
%    74138(A,B,C,G2A,G2B,G1,Y7,GND,Y6,Y5,Y4,Y3,Y2,Y1,Y0,VCC)    %

EQUATIONS:

%^% Y0=!(!A*!B*!C*!G2A*!G2B*G1);
%^% Y1=!(A*!B*!C*!G2A*!G2B*G1);
%^% Y2=!(!A*B*!C*!G2A*!G2B*G1);
%^% Y3=!(A*B*!C*!G2A*!G2B*G1);
%^% Y4=!(!A*!B*C*!G2A*!G2B*G1);
%^% Y5=!(A*!B*C*!G2A*!G2B*G1);
%^% Y6=!(!A*B*C*!G2A*!G2B*G1);
%^% Y7=!(A*B*C*!G2A*!G2B*G1);
```

292039-5

Figure 4. Network and Equations for 74138.SDF

while compiling files that use macros. The most common message is "***WARN-XLT-Node Missing Destination". This message is displayed as unused nodes from a macro are deleted. For example, if a macro using a NOCF primitive is combined with a CONF and the original feedback is not needed, the warning is displayed as the feedback is deleted.

Multiple Macro Calls

The Macro Expander allows use of more than one macro in ADFs. Each macro must have its own call, even when the same macro is used more than once.

For example, to implement two 74138s, each case or "instance" must have its own call:

```
74138(A,B,C,G2A,G2B,G1,Y7,GND,Y6,Y5,
Y4,Y3,Y2,Y1,Y0,VCC)

74138(A,B,C,G3A,G3B,G1,YF,GND,YE,YD,
YC,YB,YA,Y9,Y8,VCC)
```

In this example, many of the inputs are routed to both devices. The Macro Expander automatically generates internal nodes for each instance of the macro. Each node is assigned a unique number based on the position of the macro in the Network section (i.e., . . 0140, . . 0141, etc. for nodes connecting to the 14th primitive in the Network section).

For traceability, you can define your own instance names for nodes of different macros by including the instance name in a comment immediately following the macro call. For example, to call two 74161 macros, one as Shift Register A and the other as Shift Register B, enter the calls as follows:

```
74161(CLR,CK,A,B,C,D,ENP,,LD,ENT,QD,
QC,QB,QA,RD1,) % SFA %

74161(CLR,CK,E,F,G,H,ENP,,LD,ENT,QH,
QG,QF,QE,RC2,) % SFB %
```

The Macro Expander uses the first three ASCII characters after the first percent sign (%), except for white space, to create instance numbers. For example, internal nodes for the first three signals of each macro call will be:

```
..SFAN1, ..SFAN2, ..SFAN3,

..SFBN1, ..SFBN2, ..SFBN3
```

where SFA/SFB are the user-defined instance names and N1, N2, N3 are the node numbers associated with each instance. For cases where no internal nodes numbers are generated, the Macro Expander simply ignores the instance name.

Outputs from one macro call can be used as inputs for other calls, as follows:

```
74138(A,B,C,G2A,G2B,G1,Y7,GND,Y6,Y5,
Y4,Y3,Y2,Y1,Y0,VCC)

74138(A,B,C,Y7,G3B,G1,YF,GND,YE,YD,YC,
YB,YA,Y9,Y8,VCC)
```

Here the Y7 output from the first decoder feeds an enable input of the second decoder.

Different macros are connected in the same manner. For example, the following macro calls connect the outputs from a 74138 decoder to the inputs of 74175 latches:

```
74138(A,B,C,G2A,G2B,G1,Y7,GND,Y6,Y5,
Y4,Y3,Y2,Y1,Y0,VCC)

74175(CLR,0Q,n0Q,Y0,Y1,n1Q,1Q,GND,CLK,
2Q, n2Q,Y2,Y3,n3Q,3Q,VCC)

74175(CLR,4Q,n4Q,Y4,Y5,n5Q,5Q,GND,CLK,
6Q, n6Q,Y6,Y7,n7Q,7Q,VCC)
```

Each decoder output is routed to a 74175 input. The 74175 macro produces both true and complement latched outputs.

Guidelines/Pitfalls

The following paragraphs discuss some general guidelines for using macros:

- Because the Macro Expander supports only one level of hierarchy, there is a tendency for p-terms to multiply quickly when several macros are connected together. In many cases, the total number of p-terms exceeds the capacity of the target EPLD. One method of avoiding problems with excessive p-terms is to route the outputs from a macro function through EPLD macrocells and use the feedbacks from the macrocells as inputs to the subsequent macro functions. This partitioning of functions trades off device resources for a lower p-term count.
- Implementation of some TTL macros requires primitives that are not supported on all devices. The .DOC file for a device notes any device dependency. In many cases, a modification to the basic TTL functions results in device independence. For example, a NOCF, which is not supported on all EPLDs, can be changed to a COIF, which is supported on all devices.

- Some macros use primitives that specify an output pin (COIF, CONF, RORF, etc.). These primitives must be supported with a signal name in the OUTPUTS: declaration and by an OUTPUT call in the Network Section of the ADF. Failure to provide this support causes the following error message during compilation:

```
***ERR-XLT-undeclared output name
```

If you encounter this error, check the macro file for output primitives that require ADF support.

Macro Usage Summary

ADF macro calls must observe the following guidelines:

- Macros are called from the Network Section of an ADF.
- The name in the call must match the name in the macro file (e.g., 74138 = 74138).
- All input and output pins on the target device must have both: (1) a corresponding signal name in the INPUTS: or OUTPUTS: declaration, and (2) a corresponding INPUT or OUTPUT macro call in the Network section. It is recommended that the same node name be used on both sides of each INPUT and OUTPUT macro call. This is required when macros containing CONFs are used. (EPLD INP and CONF primitives may also be used).
- All INPUT and OUTPUT calls in the Network section must precede any other macro call.
- Node connections within an ADF are made based on the names of the nodes.
- Connections between the macro call and macro files are based on the position of signal names in the call. Therefore, the sequence of inputs and outputs in a macro call must match the sequence of inputs and outputs in the corresponding macro file.

EXAMPLE 1: TTL MACROS

This section provides an example design using TTL macros.

Circuit

The design is a two-stage decoder using a 74138 macro and two 74139 macros. Figure 5 shows the schematic for the circuit. Each 74139 macro represents one half of a TTL 74139 device. Note that two of the outputs from the 74138 are routed back to enable the two 74139 decoders.

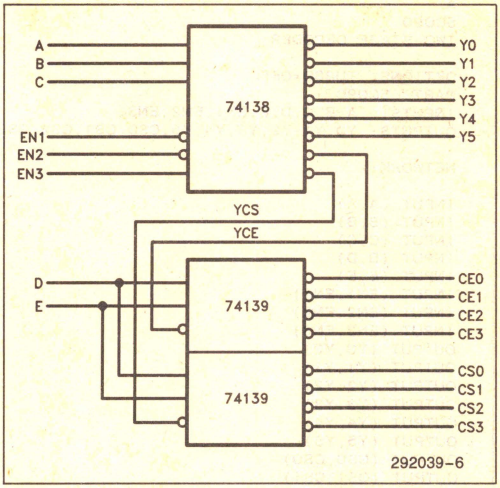


Figure 5. Schematic Diagram for Two-Stage Decoder

Figure 6 shows the ADF file containing the macro calls that implement the circuit. The two internal feedback signals (YCS and YCE) do not show up in the INPUTS: or OUTPUTS: declarations and are not represented by INPUT or OUTPUT calls in the Network section. The sequence of signals in the INPUTS: and OUTPUTS: declarations of the ADF is not important.

In the NETWORK: section, however, order is important. INPUT and OUTPUT calls must be listed before any other macro calls. This is a requirement of the Macro Expander. The sequence of signals within the ADF macro call is **critical**, as the Macro Expander automatically assigns macro call signals to macro file signals based on position.

Internal connections between macros are established by assigning the same name to the respective signals. For example, YCS in the 74138 macro call in Figure 7 represents the nY6 output from the 74138, while YCS in the 74139 macro call represents the 1G input to one 74139 decoder. Use of the same name establishes the connection. In the same manner, use of the signal name YCE connects the nY7 output from the 74138 to the 1G input of the second 74139.

```
DANIEL E. SMITH
INTEL CORPORATION
2/27/87
1
A
5C090
TWO-STAGE DECODER

OPTIONS: TURBO=OFF
PART: 5C090
INPUTS:  A,B,C,D,E,EN1,EN2,EN3
OUTPUTS: Y0,Y1,Y2,Y3,Y4,Y5,CS0,CS1,CS2,CS3,CE0,CE1,CE2,CE3

NETWORK:

INPUT (A,A)
INPUT (B,B)
INPUT (C,C)
INPUT (D,D)
INPUT (E,E)
INPUT (EN1,EN1)
INPUT (EN2,EN2)
INPUT (EN3,EN3)
OUTPUT (Y0,Y0)
OUTPUT (Y1,Y1)
OUTPUT (Y2,Y2)
OUTPUT (Y3,Y3)
OUTPUT (Y4,Y4)
OUTPUT (Y5,Y5)
OUTPUT (CS0,CS0)
OUTPUT (CS1,CS1)
OUTPUT (CS2,CS2)
OUTPUT (CS3,CS3)
OUTPUT (CE0,CE0)
OUTPUT (CE1,CE1)
OUTPUT (CE2,CE2)
OUTPUT (CE3,CE3)
74138(A,B,C,EN1,EN2,EN3,YCS,GND,YCE,Y5,Y4,Y3,Y2,Y1,Y0,VCC)
74139(YCS,D,E,CS0,CS1,CS2,CS3,GND,VCC)
74139(YCE,D,E,CE0,CE1,CE2,CE3,GND,VCC)

END$
```

292039-7

Figure 6. ADF File for Two-Stage Decoder Using TTL Macros

Sample Session

This session assumes familiarity with the iPLS II Logic Optimizing Compiler (LOC). For detailed information on the LOC, refer to Chapter 4 of the *iPLS II User's Guide,* order number: 450196. Proceed as follows to implement the TTL macro design shown here:

1. Use a standard ASCII text editor to create the ADF shown in Figure 7 under the name DECODE.ADF.
2. Invoke the iPLS II Menu by entering:

```
IPLS <Enter>
```

3. Invoke the LOC from the Main Menu by pressing <F4>.
4. Answer the LOC promts as follows:

```
Input Format?          <Enter>
File Name?             DECODE <Enter>
Minimization?          Y
Inversion Control?     N
LEF Analysis?          Y
Error Message File     <Enter>
```

The LOC then asks:

```
Do you wish to run under the above
conditions [Y/N]?
Enter: Y
```

The LOC expands the macros and compiles the expanded file to produce a JEDEC programming file (DECODE.JED), a utilization report file (DECODE. RPT), a minimized equation file (DECODE. LEF), and an error message file (DECODE.ERR). For tracability, a file called DECODE.SDF is created to show the expanded form of the ADF output by the Macro Expander.

5. The LOC terminates execution with the following message:

```
LOC cycle successfully completed
```

You can examine the LEF file to see the minimized form of the design. The LEF shows the EPLD primitives used to implement the design. Macro calls are not shown. If you wish, you can also use LPS (Logic Programmer Software) to program a part.

EXAMPLE 2: GATE-ARRAY MACRO

This section shows an example design using gate-array macros.

Circuit

The design is a two-bit adder. Figure 7 shows the schematic for the target circuit. Figure 8 lists the gate array macro file for a single-bit full adder. Figure 9 shows the ADF for the target design that includes two instances of the single-bit adder macro call. Each instance includes a user-defined instance name in the comment after the call (BT0 and BT1). These instance names will be used to identify internal nodes, if the ADRF macro contains internal nodes. Note that inputs call the PTIN (TTL Receiver) macro while outputs call the PCOWP2 (2 mA CMOS Push Pull Driver) macro. During macro expansion, an EPLD INP primitive is substituted for the gate array PTIN macro and an EPLD CONF primitive is substituted for the PCOWP2 macro. This illustrates one of the differences between TTL and gate array macros.

Once again, the connections between the two macros is dependent on the position of signals in the call. CARRY1 from the first instance of the ADRF macro connects to CARRY1 in the second instance of the macro. This corresponds to the CYOUT (Carry Out) from the adder for the first bit feeding the C0 (or Carry In) input of the adder for the second bit. Note that the C0 input of the adder for the first bit is not connected to any node in the macro call; in this case, the default value from the macro file (GND) is used to disable the input (No Carry).

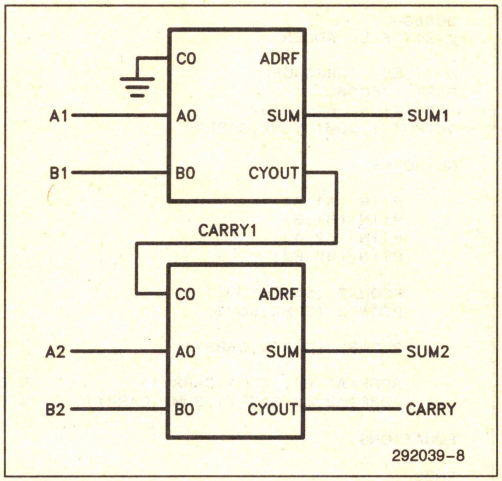


Figure 7. Schematic for Two-Bit Adder

Sample Session

To implement this ADF in an actual session, follow the steps described for Example 1, substituting the name ADDER2 for DECODE. iPLS II produces a JEDEC programming file (ADDER2.JED), a utilization report file (ADDER2.RPT), a minimized equation file (ADDER2.LEF), and an error message file (ADDER2.ERR). For traceability, a file called ADDER2.SDF is created to show the expanded form of the ADF output by the Macro Expander.

```
ADRF(A0,B0,C0,SOUT,CYOUT)
DEFAULT:(GND,GND,GND,,)

%FULL ADDER%

NETWORK:

EQUATIONS:

SOUT = A0' * B0' * C0
     + A0' * B0 * C0'
     + A0 * B0' * C0'
     + A0 * B0 * C0;
CYOUT = A0 * B0
      + B0 * C0
      + A0 * C0;

ENDEF
```

292039-9

Figure 8. Macro File for Full-Bit Adder

```
ROGER AUBLE
INTEL CORPORATION
2/27/87
1
A
5C060
2-BIT FULL ADDER

OPTIONS: TURBO=OFF
PART:  5C060
INPUTS: A1,B1,A2,B2
OUTPUTS: SUM1,SUM2,CARRY

NETWORK:

      PTIN (A1,A1)
      PTIN (B1,B1)
      PTIN (A2,A2)
      PTIN (B2,B2)

      PCOWP2 (SUM1,SUM1)
      PCOWP2 (SUM2,SUM2)

      PCOWP2 (CARRY,CARRY)

      ADRF(A1,B1,,SUM1,CARRY1)          % BT0 %
      ADRF(A2,B2,CARRY1,SUM2,CARRY)     % BT1 %

EQUATIONS:

END$
```

292039-10

Figure 9. ADF For Two-Bit Adder Using Gate-Array Macros

EXAMPLE 3: MIXING MACROS AND EPLD PRIMITIVES

This final example uses TTL macros together with standard EPLD primitives.

Circuit

The example circuit here is the 74138 macro used in example 1 with two of the outputs routed through additional combinatorial logic and RONF (Registered Output — No Feedback) primitives. Figure 10 shows the circuit. CS2 and CS3 are qualified by two additional inputs (RD* and WR*) to set or clear two latches. This is a configuration commonly used in microcomputer systems, where control signals are set and reset based on the address and command signals but not on a data value. A read to the port decoded by CS2 sets output LCS2 (Latched CS2) high. A write to that same port clears LCS2 low.

Figure 11 shows the ADF that implements the example circuit. This is the same ADF used in Figure 6, with the addition of several primitives and equations. The data inputs to both latches are tied to VCC. When RD* and the chip enable are both low, the respective clock signal goes low. As RD* or chip enable go high, the rising edge of the clock signal triggers the register, driving the output high.

Note that many Intel EPLDs do not support multiple product terms for register clocks. Therefore, the clock buffer primitive is driven by a macrocell configured as a COIF (Combinatorial Output—Input Feedback). Control signals (Clear and Preset) for many EPLDs also support only one product term. In this case, however, the NOR gate driving the clear input to the RONFs can be minimized to a single p-term. Thus a low on WR* and chip enable clears the respective latch to logic 0. (The intermediate macrocell for the Read function can be omitted for EPLDs that support two p-terms on register clocks.)

The connections between the TTL macros and the EPLD primitive are made by assigning the appropriate names to the input and output nodes. The CS2 and CS3 signals from the first example are no longer outputs, but are simply inputs to equations that feed the LCS2 and LCS3 RONF primitives.

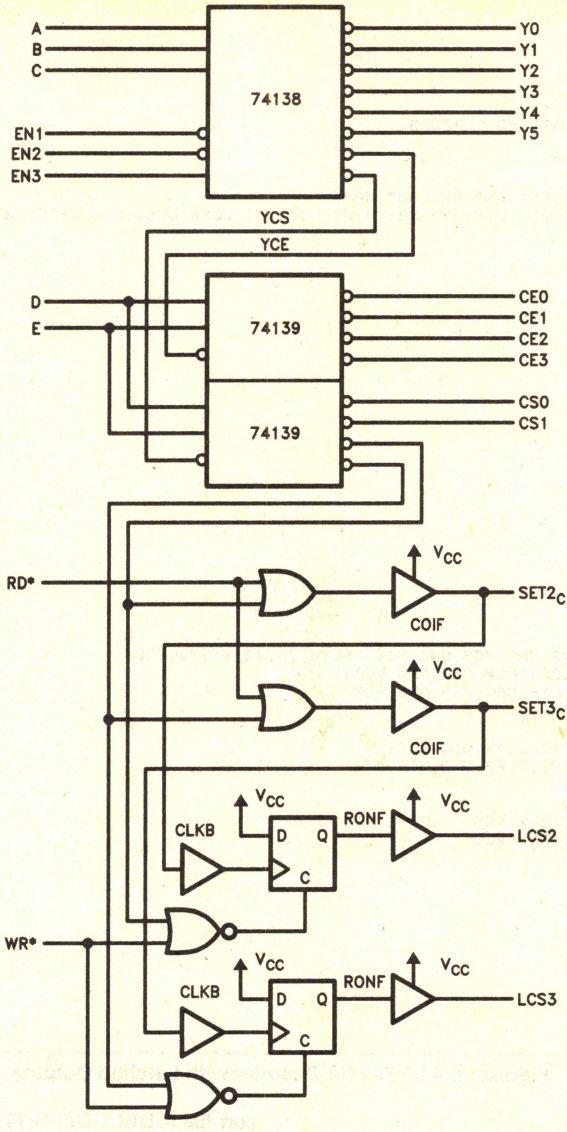


Figure 10. Schematic of Decoder Circuit with Latched Outputs

```
DANIEL E. SMITH
INTEL CORPORATION
2/27/87
1
A
5C090
DECODER WITH TWO LATCHED OUTPUTS

OPTIONS:   TURBO=OFF
PART:   5C090
INPUTS:   A,B,C,D,E,EN1,EN2,EN3,RD*,WR*
OUTPUTS:   SET2c,SET3c,Y0,Y1,Y2,Y3,Y4,Y5,CS0,CS1,LCS2,LCS3,CE0,CE1,CE2,CE3

NETWORK:

INPUT (A,A)
INPUT (B,B)
INPUT (C,C)
INPUT (D,D)
INPUT (E,E)
INPUT (EN1,EN1)
INPUT (EN2,EN2)
INPUT (EN3,EN3)
OUTPUT (Y0,Y0)
OUTPUT (Y1,Y1)
OUTPUT (Y2,Y2)
OUTPUT (Y3,Y3)
OUTPUT (Y4,Y4)
OUTPUT (Y5,Y5)
OUTPUT (CS0,CS0)
OUTPUT (CS1,CS1)
OUTPUT (CE0,CE0)
OUTPUT (CE1,CE1)
OUTPUT (CE2,CE2)
OUTPUT (CE3,CE3)
74138(A,B,C,EN1,EN2,EN3,YCS,GND,YCE,Y5,Y4,Y3,Y2,Y1,Y0,VCC)
74139(YCS,D,E,CS0,CS1,CS2,CS3,GND,VCC)
74139(YCE,D,E,CE0,CE1,CE2,CE3,GND,VCC)
RD = INP(RD*)
WR = INP(WR*)
LCS2 = RONF(VCC,SET2,CLR2,GND,VCC)
LCS3 = RONF(VCC,SET3,CLR3,GND,VCC)
SET2 = CLKB(SET2c)
SET3 = CLKB(SET3c)
SET2c,SET2c = COIF(ST2,VCC)
SET3c,SET3c = COIF(ST3,VCC)

EQUATIONS:

ST2 = RD + CS2;
CLR2 = /(WR + CS2);
ST3 = RD + CS3;
CLR3 = /(WR + CS3);

END$
```

292039-12

Figure 11. ADF File for Decoder with Latched Outputs

Sample Session

To implement this ADF in an actual session, follow the steps described for Example 1, substituting the name LDECODE for DECODE. iPLS II produces a JEDEC programming file (LDECODE.JED), a utilization report file (LDECODE.RPT), a minimized equation file (LDECODE.LEF), and an error message file (LDECODE.ERR). For traceability, a file called LDECODE.SDF is created to show the expanded form of the ADF output by the Macro Expander.

intel

APPLICATION NOTE

AP-312

June 1987

Creating Macros for EPLD Designs

DANIEL E. SMITH
APPLICATIONS ENGINEERING

Order Number: 292040-001

INTRODUCTION

The iPLS II (Intel Programmable Logic Software II) Logic Optimizing Compiler includes a Macro Expander that supports the use of macros in EPLD designs. These macros can include TTL and Gate Array macros available from Intel, or proprietary macros developed by a user. This application note shows how to create user-defined macros and how to build macro libraries with Intel's Macro Librarian, an optional software package for use with iPLS II. A design example also shows creation of a user-defined macro and its use in an ADF (Advanced Design File). Detailed information on using the TTL and Gate Array Macros in iPLS II ADFs are described in a companion application note, AP-311 "Using Macros in EPLD Designs", Order Number: 292039. This application note concentrates on creating macros; it assumes that you have read and understood the discussion on using macros in AP-311.

OVERVIEW

iPLS II allows designers to include macro calls in design files to implement common circuit functions. Macros calls are subsequently expanded by the LOC (Logic Optimizing Compiler) into the ADF network and/or equation entries required to perform the desired functions. Macros can be connected together or used in conjunction with standard iPLS II EPLD primitives.

By following the macro file format described in this note, users can also create their own proprietary macros with an ASCII text editor. These macro files can then be stored in user-defined libraries by using Intel's Macro Librarian software. User-defined macros can be called from ADFs created by a text editor or by schematic capture software that supports user-defined symbols and that outputs in ADF format. User-defined macros can optimize development of EPLD designs by modularizing the design process and by allowing the design process to proceed at a higher level than with EPLD primitives alone. iPLS II support for user-defined macros (see in Figure 1) includes the following:

- MLIB, the optional iPLS II Macro Librarian for creating macro libraries from individual user-defined macro files.
- a Macro Expander in the LOC that expands macro calls in ADFs with the contents of the corresponding macros from libraries.

This application note describes how to create macro files, store them in libraries with MLIB, and shows how to call them from ADFs created by a text editor. For information on creating user-defined macro symbols with schematic capture packages, refer to the appropriate manual for the schematic capture package you are using. SCHEMA II-PLD available from Intel supports user-defined symbols and outputs in ADF format.

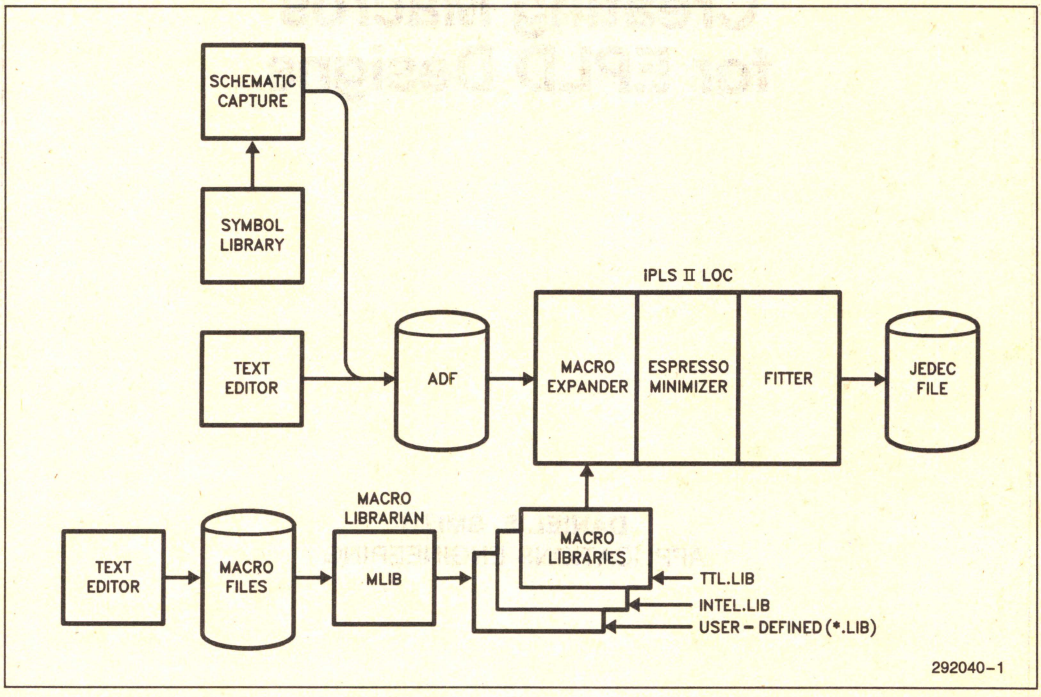


Figure 1. Macro Support for iPLS II

(SCHEMA II-PLD is based on SCHEMA II from Omation, Inc. The Intel EPLD Design Manager, also available from Intel, allows existing SCHEMA II users to design with EPLDs and macros.)

MACRO FILES

This section describes iPLS II macro files. **User-defined macro files must follow the guidelines presented here** to be successfully processed by the Macro Librarian (MLIB) and expanded by the iPLS II LOC Macro Expander.

Macro filenames follow DOS conventions. It is recommended that macro filenames end with the extension .DEV, which is the default for MLIB. Only one macro can be contained in a macro file. Macro files are comprised of three sections:

- Header
- Network Section
- Equation Section

All macro files must end with the literal "ENDEF". Figure 2 shows a sample macro file for a proprietary part (16207), a "black box" containing random logic.

```
16207(A,B,C,D,E,F,U,V,W,X,Y,Z)
DEFAULT:(GND,GND,GND,VCC,VCC,VCC,,,,,,)

EQUATIONS:
        U = /(A * B);
        V = /(/E * A * B);
        W = /(D * C * A * /E);
        X = /(/D * E);
        Y = /(F * D * A);
        Z = F * /E;

ENDEF
                                    292040-2
```

Figure 2. Sample Macro File for "Black Box" (16207.DEV)

Header

Headers for macro files contain two lines. The first line includes the name of the macro function and a list of inputs and outputs for the macro. The second line contains defaults for the device.

The name of the macro can be a device number (16207, 83546, etc.), function name (ADDRCNT, CMDLO, etc.), or any name up to eight characters long. No spaces or comments precede the name.

Inputs and Outputs follow immediately after the macro name and are enclosed in parentheses. I/O signal names may be up to eight characters long, but may not contain pin numbers. For user-defined macros, signals may be listed in any order desired. For example, any of the following entries are legal:

16207 (A,B,C,D,E,F,U,V,W,X,Y,Z)

16207 (B,D,A,R,Z,U,W,C,F,X,E,Y)

16207 (Z,Y,X,W,V,U,F,E,D,C,B,A)

Note that this **first line of the header forms the template used to call the Macro from the ADF.** The Macro Expander connects ADF nodes in the macro call to I/O signals in the macro file on the basis of **position**, not on the basis of node name.

The second line in the header specifies defaults for inputs (VCC or GND) in cases where those signals are left unconnected. The DEFAULT: line must be included in the macro definition file, even when no defaults are used in the ADF. The keyword DEFAULT: is the first entry in this line. The default values for all signals follow immediately and are enclosed in parentheses. Input defaults may be VCC or GND. The position of the default value corresponds to the signal listed in the previous line.

Defaults for outputs are blank, but a comma (,) must be present (place holder) for each output signal except the last. For example, the 16207 black box contains six inputs (A through F) and six outputs (U through Z). The first two lines for this macro might be:

```
16207 (A,B,C,D,E,F,U,V,W,X,Y,Z)
DEFAULT: (GND,GND,GND,VCC,VCC,VCC,,,,,,)
```

Defaults for inputs A through C are GND; defaults for inputs D through F are VCC. Defaults for the outputs are not specified, but the comma denotes the positions for those signals.

Defaults should be chosen with care. Clears, Presets, Loads, etc. should be disabled in most cases. Enables should be enabled. Input defaults can also be left blank as long as those inputs are connected to nodes in the ADF that calls the macro, but it is recommended that they be specified in the macro file.

Network Section

The NETWORK: section lists the EPLD primitives used to implement the desired functions. The Network Section follows ADF syntax rules. As far as possible, the macros should be implemented in equations to eliminate concern about feedbacks and output enables. In the case of a circuit that requires macrocell registers, the feeback-only form of the primitive should be used so that the Macro Expander can make the correct pin connections. The following example shows this:

```
OUT1 = NORF (INd,CLK,GND,GND)
```

During processing, the Macro Expander connects the feedback to an output (if necessary) and supplies the required output enable node name. The Macro Expander also eliminates unneeded Network and Equations entries if they are not used by an ADF.

If no network entries are required (i.e., a macro implemented entirely in equations), the entire Network section may be omitted, including the keyword NETWORK:. In many cases, equations alone can implement the desired functions.

Equations Section

The EQUATIONS: section lists the Boolean equations for the desired functions and follows ADF syntax rules, with one exception; intermediate equations are not permitted in macro files. If no equation entries are required (i.e., a macro implemented entirely in the Network Section), the entire Equation section may be omitted, including the keyword EQUATIONS:.

Comments and White Space

Comments can be placed anywhere in a macro file except before the name and signals on the first line. Comments must be enclosed in percent signs, as follows:

```
% THIS IS A SAMPLE COMMENT %
```

White space can appear on any line except the first two lines.

MACRO LIBRARIAN

The Macro Librarian (MLIB) is an optional software package that combines individual macro files into macro libraries. These libraries are in turn used by the LOC Macro Expander. MLIB can be invoked from the command line, from command files, or from a combination of both. Figure 3 shows a block diagram of the Macro Librarian.

Syntax for MLIB command lines is as follows:

MLIB [-options] [@cmdfile] [file1 file2 . . .]
<Enter>

-d directory. Displays directory information for the library being created.

-v verbose. Print status during processing. When not specified, status messages are suppressed.

-l lib list. Lists the contents of existing macro library to console. This option may not be used while building a library.

-o lib name of the target macro library. MACRO.LIB is the default when no name is specified. TTL.LIB and INTEL.LIB are reserved for Intel libraries and may not be used.

-s string include version stamp in macro library. The version string can be up to 7 characters long. "V1.00" is the default stamp.

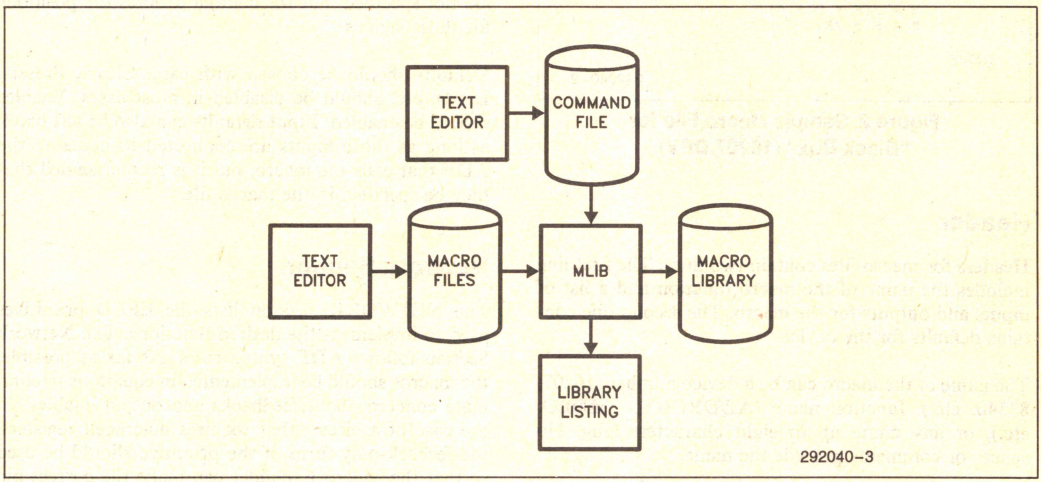


Figure 3. Macro Librarian Block Diagram

-c string include copyright string in macro library. The copyright string can be up to 61 characters long and, if blanks are used, must be contained in quotation marks, for example, "texta textb".

@cmdfile name of command file. The command file can include options and macro filenames. The @ symbol must precede the filename.

file1 . . . name of device files to be included in the macro library. Separate files by spaces.

For example, the following command line:

MLIB -v -s 2.00 -o USER.LIB @USERLIST <Enter>

creates a library called USER.LIB that includes all the individual macro files contained in the command file USERLIST. MLIB displays status messages as it processes the macro files in USERLIST (-v). The library is created as version 2.00 (-s).

Macro library filenames follows DOS conventions and should end with the extension .LIB to be recognized by the Macro Expander. INTEL.LIB and TTL.LIB are reserved and may not be used.

USERLIST is the name of the command file and must be preceded by the @ symbol. The command file is simply an ASCII text file that can be modified to contain any number of macros desired. MLIB processes the entire list of macros on each invocation. To add a new macro to an existing library, add the name of the macro to USERLIST, and create the new library by entering the command line shown above. Command file names follow DOS conventions. MLIB supplies a .DEV extension if no extension is specified. MLIB searches first in the current directory, then along the DEV environment variable, and finally along the PATH environment variable for the files.

In order to connect inut and output primitives, the files INPUT.DEV and OUTPUT.DEV must be included in at least one of the libraries. These files are contained in the TTL and Intel Gate Array macro libraries (TTL.LIB and INTEL.LIB, respectively).

Figure 4 shows a sample MLIB command file that includes options, the library name, and the names of seven macro files to be included in the library in addition to the INPUT and OUTPUT macros. The format of the command file is free form. Note that comments can be included in the command file and must be contained within percent (%) signs.

Note that the -1 option cannot be included in an MLIB command file; it can only appear on the command line. The -1 option lists the contents of existing libraries; it does not list library contents while building a library.

```
-o PROJA.LIB % macro library name %
-v
-s V1.50 % version number %
-c "Copyright  (C) 'Date, Your Company, Your Name"
                % copyright information %
-d % display directory %

% include the following macros %

INPUT.DEV    OUTPUT.DEV    7408.DEV
7487.DEV     74138.DEV     74139.DEV
74151.DEV    74157.DEV     74251.DEV
                                        292040-4
```

Figure 4. Sample Command File for MLIB

The command line to process the file shown in Figure 4 is as follows:

MLIB @SAMPLE <Enter>

where SAMPLE is the name of the command file.

To list the contents of PROJA.LIB after creation, invoke MLIB as follows:

MLIB -1 PROJA.LIB

This command line lists the macros in PROJA.LIB to the screen. The DOS file redirection capability can also be used to create a disk file listing the contents of macro libraries. For example:

MLIB -1 PROJA.LIB > PROJA.DOC

SAMPLE SESSION: COMMAND DECODER USING MACROS

Decoding logic is one common function implemented by programmable logic devices. The target circuit for this example is a device that decodes microprocessor command signals in selected address ranges. The target application and decoder requirements are as follows:

- The target application is a 16-bit microcomputer system with 1-Megabyte of memory and about two dozen I/O ports.
- The memory is divided into shared memory (lower 512K bytes) and local memory (upper 512K bytes). Shared memory resides off the processor board and requires active low memory command signals. Local memory resides on-board and requires active high memory command signals.
- I/O ports are also split between on-board devices requiring active high signals and off-board devices requiring active low signals. I/O devices between the address range F000-FFFFH are on-board; devices below that range (0000-EFFFH) are off-board.

- All interrupt requests are resolved by an on-board interrupt controller. Therefore, only an active high on-board interrupt acknowledge signal is needed.
- On-board control signals are always high or low, never three-stated. Off-board control signals are three-stated when not being used to execute a bus cycle. An external bus arbiter accepts a request signal from the command decoder and, after gaining control of the bus, sends address enable and command enable signals back to the command decoder.

Figure 5 shows a block diagram of the application, including the target EPLD design. The three functional blocks to be included in the EPLD are highlighted (not shaded).

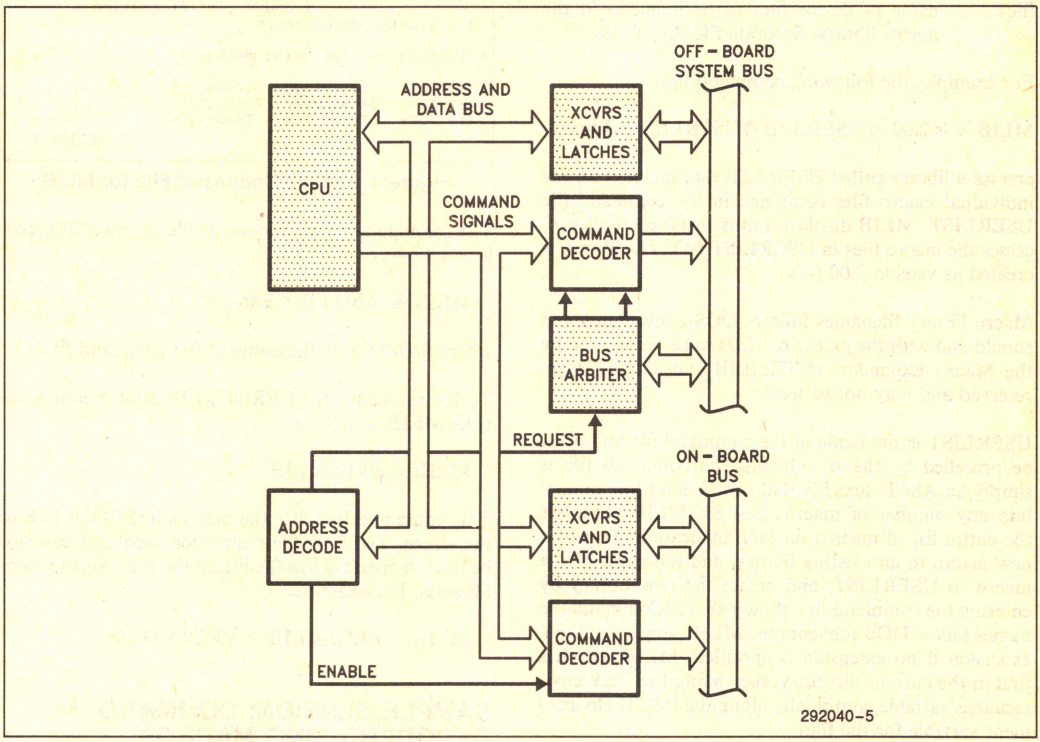


Figure 5. Block Diagram of Target Circuit and Application

Creating the Macro

Figure 6 shows a schematic diagram for the active low command decoder implemented with OR gates (low inputs enable the outputs; high inputs disable the outputs). Figure 7 shows the macro file that implements the circuit (CMDLO.DEV). This file was created with an ASCII text editor. Used as is, it provides the active low outputs for the design. With inputs RD, WR, and INTAIN inverted, it also provides the active high outputs for the design. This design uses CONF primitives to implement the three-state outputs in the macro. As an alternative, equations alone could have been used with the CONFs included in the ADF.

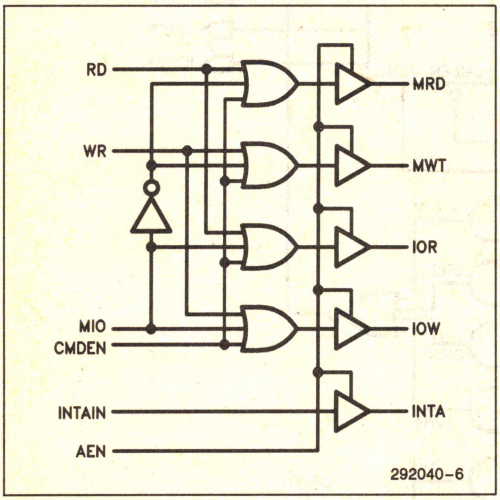


Figure 6. Schematic Diagram of Command Decoder

Building the Library

Use your text editor to create an MLIB command file that includes CMDLO.DEV, INPUT.DEV, and OUTPUT.DEV. The following example shows a sample command file named MACLIST.

```
-v  % show status %
-c  "1987, AP-312 Sample Macro Library"
-o  AP312.LIB
-d  % show the list %

% include the following macros %

CMDLO.DEV  INPUT.DEV  OUTPUT.DEV
```

Invoke the Macro Librarian with the following command line:

```
MLIB  @MACLIST
```

The Macro Librarian processes the three macro files and stores them in a user library named AP312.LIB. The library contains the copyright statement "1987, AP-312 Sample Macro Library". When processing is complete, MLIB returns control to DOS.

Creating the ADF

Figure 8 shows a schematic diagram for the target circuit. Figure 9 shows the ADF for the circuit (COMCODE.ADF), which invokes both instances of the CMDLO macro and contains equations used to enable the decoders under the proper conditions. The ADF signal named ONBEN (On-Board Enable) enables the active high decoder). The AEN (Address Enable) input to the on-board decoder is left unconnected. The default (always enabled) will be used.

```
CMDLO(MIO,RD,WR,INTAIN,CMDEN,AEN,MRD,MWT,IOR,IOW,INTA)
DEFAULT:(GND,VCC,VCC,VCC,GND,GND,,,,,)

NETWORK:

MRD  = CONF(MRDc,AEN)
MWT  = CONF(MWTc,AEN)
IOR  = CONF(IORc,AEN)
IOW  = CONF(IOWc,AEN)
INTA = CONF(INTAIN,AEN)

EQUATIONS:

MRDc = /MIO + RD + CMDEN;
MWTc = /MIO + WR + CMDEN;
IORc = MIO + RD  + CMDEN;
IOWc = MIO + WR  + CMDEN;

ENDEF
```

292040-7

Figure 7. Macro File for Command Decoder (CMDLO.DEV)

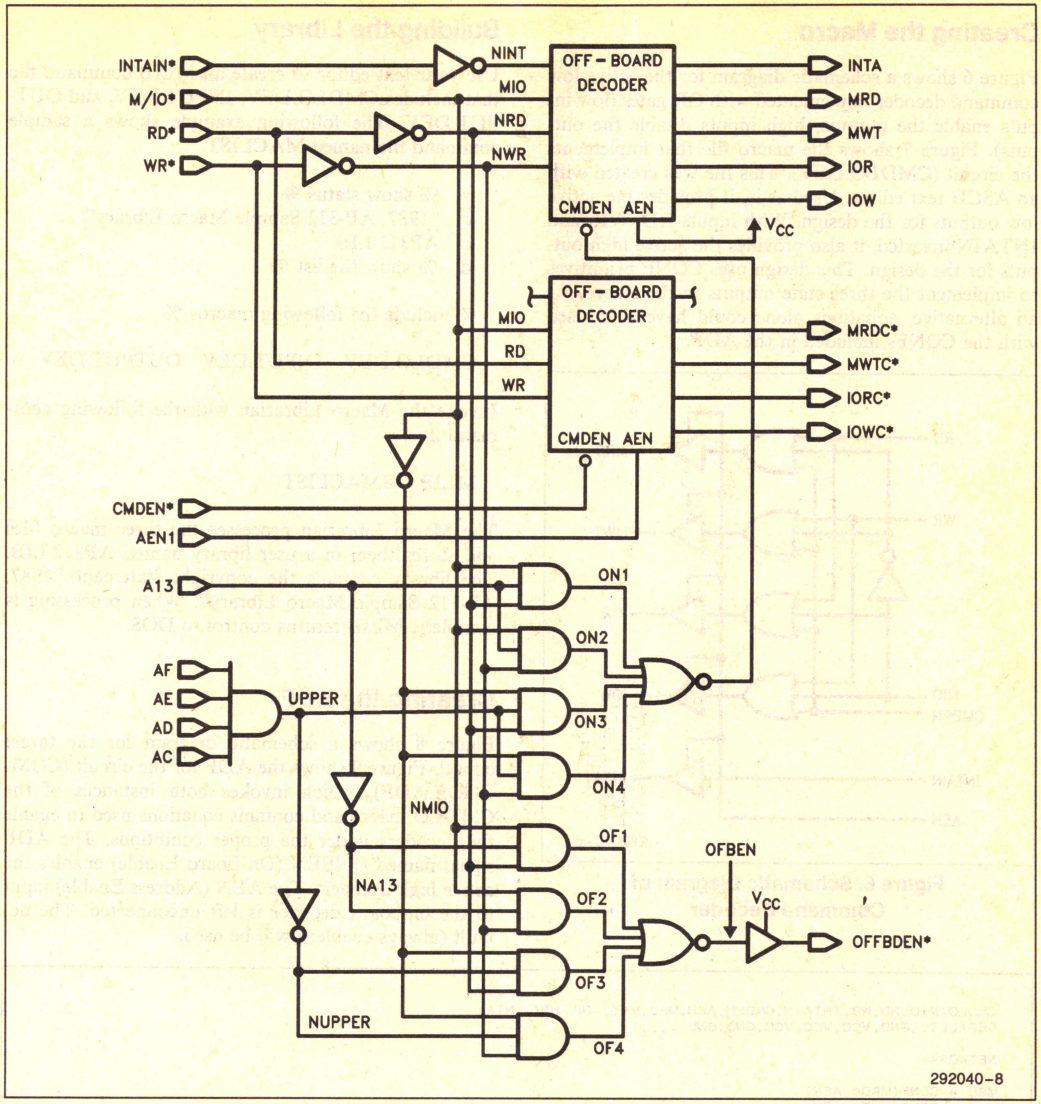


Figure 8. Schematic Diagram for COMCODE.ADF

```
DANIEL E. SMITH
INTEL CORPORATION
4/7/87
1
A
16209-001
COMMAND DECODER

OPTIONS:  TURBO=ON
PART:  5C090
INPUTS:  MIO, RD, WR, INTAIN, CMDEN, AEN1, A13, AF, AE, AD, AC
OUTPUTS:  MRD, MWT, IOR, IOW, INTA, MRDC, MWTC, IORC, IOWC, OFFBDEN

NETWORK:

INPUT(MIO,MIO)
INPUT(RD,RD)
INPUT(WR,WR)
INPUT(INTAIN,INTAIN)
INPUT(CMDEN,CMDEN)
INPUT(AEN1,AEN1)
INPUT(A13,A13)
INPUT(AF,AF)
INPUT(AE,AE)
INPUT(AD,AD)
INPUT(AC,AC)

OUTPUT(MRD,MRD)
OUTPUT(MWT,MWT)
OUTPUT(IOR,IOR)
OUTPUT(IOW,IOW)
OUTPUT(INTA,INTA)
OUTPUT(MRDC,MRDC)
OUTPUT(MWTC,MWTC)
OUTPUT(IORC,IORC)
OUTPUT(IOWC,IOWC)

CMDLO(MIO,RD,WR,,CMDEN,AEN1,MRDC,MWTC,IORC,IOWC,) % OFB %

CMDLO(MIO,NRD,NWR,NINT,ONBEN,VCC,MRD,MWT,IOR,IOW,INTA) % ONB %

OFFBDEN = CONF(OFBEN,VCC)
OFBEN   = NOR(OF1,OF2,OF3,OF4)
ONBEN   = NOR(ON1,ON2,ON3,ON4)
NRD     = NOT(RD)
NWR     = NOT(WR)
NINT    = NOT(INTAIN)
NMIO    = NOT(MIO)
NUPPER  = NOT(UPPER)
NA13    = NOT(A13)

EQUATIONS:

UPPER = (AF * AE * AD * AC);
ON1   = (MIO * A13 * NRD);
ON2   = (MIO * A13 * NWR);
ON3   = (NMIO * UPPER * NRD);
ON4   = (NMIO * UPPER * NWR);
OF1   = (MIO * NA13 * NRD);
OF2   = (MIO * NA13 * NWR);
OF3   = (NMIO * NUPPER * NRD);
OF4   = (NMIO * NUPPER * NWR);

END$
```

292040-9

Figure 9. ADF for COMCODE.ADF

OFFBEN (Off-Board Enable) requests permission to access the off-board bus from the external bus arbiter. The bus arbiter enables the off-board decoder via AEN1 (Address Enable 1) and CMDEN (Command Enable). CMDEN allows the appropriate signal to go high or low, and AEN1 causes the outputs to independently enter or exit a high impedance state (three-state).

Note the same name is used for both nodes of each INPUT and OUTPUT macro call. Use of the same name ensures proper connection when the Macro Expander eliminates redundant primitives (for example, a CONF feeding another CONF).

Compiling the Design

Proceed as follows to compile the ADF.

1. Include AP312.LIB in the IPLS environment variable. From the DOS command prompt, type:

   SET IPLS=C:\IPLSII\AP312.LIB; . . . <Enter>

   For user-defined macro libraries that are regularly accessed, the IPLS variable can be set in an AUTOEXEC.BAT file.

2. Invoke the iPLS II Menu by entering:

   IPLS <Enter>

3. Invoke the LOC from the Main Menu by pressing <F4>.

4. Answer the LOC prompts as follows:

Input Format?	<Enter>
File Name?	COMCODE <Enter>
Minimization?	Y
Inversion Control?	N
LEF Analysis?	Y
Error Message File	COMCODE.ERR<Enter>

   The LOC then asks:

   Do you wish to run under the above conditions [Y/N]?

   Enter: Y

   The LOC expands the macros and compiles the expanded file to produce a JEDEC programming file (COMCODE.JED), a utilization report file (COMCODE.RPT), a minimized logic equation file (COMCODE.LEF) and an error message file (COMCODE.ERR). For traceability, a file called COMCODE.SDF is created to show the expanded form of the ADF output by the Macro Expander.

5. The LOC terminates execution with the following message:

   LOC cycle successfully completed

You can examine the LEF file to see the minimized form of the design. The LEF shows the EPLD primitives used to implement the design. Macro calls are not shown in the LEF. If you wish, you can also use LPS (Logic Programmer Software) to program a part.

Tools for Optimizing PLD Designs

Alan J. Coppola
Tool Architect

Intel Corporation
M/S EY2-11
5200 NE Elam Young Pkwy.
Hillsboro, OR 97123
(503)681-2177

Introduction:

The purpose of this paper is to describe a design methodology for Programmable Logic Devices(PLD's) and to survey current PLD optimization techniques.

1. Perspective: Where do PLD's fit in?

The use of Programmable Logic Devices(PLD's) represents a middle ground in logic design. The two common approaches to logic implementation in today's market are Board Design methods(building a solution from a selection of pre-fabricated standard parts - TTL/SSI/MSI) and Custom/Semi-Custom design methods(fabricating a custom logic chip to solve the problem at hand, and then building a much simpler board).

With the Board Design approach, PCB's carry the fruit of a designer's labor to the customer. Many little black boxes and other electrical circuit components make up the brunt of a PCB's load. Many times there are large islands of functionality to be connected together via encoding/decoding and timing circuits. The islands of functionality (i.e. microprocessor, microcontroller, RAM, EPROM, transciever, etc.) all have different protocols, and all speak different languages at different speeds.

Integrating the major devices of a board together involves much "glue" logic. The typical designer spends time and effort looking through a TTL parts catalog to find the best fit for a design based on functionality, performance and price. After a preliminary function-based board is laid out, modification passes are made based on the parts needed and their availability. Large designs increase the length and risk of this process. Even though most major CAD vendors are addressing the problem of board design, simulation, test and interface with the Custom/Semi-Custom arena, board design and manufacturing tools are rapidly becoming the primary practical obstacle to effective production of end-user systems. PLD's reduce the complexity of the end-user board, hence reducing the length and risk of the implementation and manufacturing process.

With the Custom/Semi-Custom design approach, the designer is free to address functionality directly. The designer has much flexibility in the functionality, speed and integration facets of logic design. Certainly, the number of parts on an end-user PCB can be greatly reduced by integrating most of a board's function into a few custom devices. The problems associated with all the flexibility lie in the physical design, modeling, and tools areas. Specifically, in the physical design area, problems include process specific bottlenecks, NRE charges and long lead times. In the modeling area, extensive simulations must occur before and after the device is built, as each device is custom crafted. Finally, in the CAD tools area, a highly functional, but hard to use set of tools guide and control the whole process. The tools are the best in terms of functionality, but the worst in terms of cost and ease of use.

The job of ASIC vendors in the next ten years is to make the Custom/Semi-Custom problems disappear or become acceptable to the logic designer of the next generation. The facts are clear. Without advanced tools which automate much of the logic designer's work, the Custom/Semi-Custom approach only works for large scale, large volume or special purpose devices. Silicon compilers and other Custom/Semi-Custom design methodologies are working hard to overcome the inherent problems of this type of design.

The use of PLD's in a design is a compromise between the flexibility of a Custom/Semi-Custom design, and the standard Board design methodolgy.

The definition of PLD which I am using for the purposes of this paper is very general. A Programmable Logic Device is any device, which can be programmed by the user, to realize a chunk of combinatorial or sequential logic. A subset of the most popular, or newest types of PLD's are: PAL's, PLE's(Monolithic Memories), EPLD's(Intel, Altera), EEPLD's(Lattice), FPLA's, FPLS's(Signetics), LCA's(Xilinx), and ERASIC's(Exel). All except the last two are based on some form of two-level(AND/OR) registered array logic. I will mainly be concerned with two-level array logic devices.

The good points of designing with PLD's are:

1. The integration of many small chunks of TTL and SSI/MSI logic into a few PLDs. Essential for efficient use of resources.

2. An easier overall development cycle than the Custom/Semi-Custom route. Also easier than standard board TTL development cycle once the learning curve is passed.

3. Much cheaper than the Custom/Semi-Custom design method for all but large volume designs. Usually cheaper than standard board TTL design method.

4. The breadboard character and modifiability of PLDs makes them an excellent R&D and learning vehicle in the design environment.

5. PLD CAD development tools are available to convert standard TTL logic representations into the complicated fixed architectures of an individual device. The CAD tools automate this process, to a large degree, so that the user can use their own design techniques.

The bad points of designing with PLDs' are:

1. For large designs, using PLD's is not as functional or robust as using Custom/Semi-Custom logic.

2. The CAD tools available for PLD design are not as useful in automating the whole design process as those in the Custom/Semi-Custom arena. In fact, most of the tools are derived from those used in Custom/Semi-Custom design.

3. The speed and function constraints of the fixed device architectures can be inhibiting. For example, not all types of designs fit well into two-level array logic.

2. The PLD Development Environment

There are three pieces in a PLD development environment. First is the input part. The design specification needs to be entered in some form, such as a schematic, a finite state machine, or a high level language description. Second is the processing part. This must include some kind of compilation of the input into object code(JEDEC code), and usually also includes optimization of the design. Third is the output part. This includes the object code(JEDEC), test results/vectors, and statistics.

A PLD development environment has an underlying language for representing design specifications, which is usually more general than the intended devices. Often, the system provides means for accepting input in other forms, which then translates into the underlying language. We shall use the standard term "Hardware Description Language"(HDL) when refering to this language, as this is where these languages are heading in terms of complexity and future directions.Examples of PLD HDLs' are ABEL by Data/IO-Futurenet, CUPL by Assisted Technology, ADF by Intel(Altera), PALASM by Monolithic Memories, LOG/IC by Kontron, and AMAZE by Signetics.

There are four questions one can ask about a PLD development environment and it's HDL. The answers to these questions determine, for the user, which systems to use. The four questions are:

Question 1: Is it easy to learn and easy to use(User-Friendly) ?

Question 2: Is it generic enough to support the current applications and new devices yet to come ? (HDL expressiveness and functionality)

Question 3: Does the compiler use optimization techniques to produce JEDEC code ? (i.e. logic minimization)

Question 4: Are there alternate logic entry tools, like schematic capture and Finte State Machine entry, are there hooks for simulation and other design methodologies, like gate arrays ?

As an example, we answer the four questions for the HDL of the Intel Programmable Logic Development System(iPLDS)(Altera). The HDL of iPLDS is called Advanced Design File(ADF).

Question 1: Is it easy to learn and easy to use ?(User-Friendly)
Answer: The ADF language is made easier to learn for the novice user by two items:

a. Graphical Interface Tools

Logic Builder(LB): A graphical netlist entry and syntax checking aid to the user entering designs which have already been written down on paper in a schematic fashion.

Logic Programming(LP): A graphical JEDEC file editor, which allows the user to modify the JEDEC file. More importantly, the tool allows the user to investigate and learn the device architecture via a user-oriented graphical interface, and then to program the part directly from this interface.

b. Primitives:
A large(~80) set of logic and I/O Macrocell primitives which capture all of the current standard ways to represent small chunks of memory and combinational logic. This is useful for the novice and occasional user, who doesn't have the time, or want to learn the abstractions involved in more generic HDLs'.

Question 2: Is it generic enough to support the current applications and new devices yet to come ? (HDL expressiveness and functionality)
Answer: Yes, it is generic enough to support those current and future architectures based on two-level registered logic, which are produced by Intel and Altera. The large number of primitives make the language unwieldly for support of new devices not falling into this realm. Finally, it supports only Intel and Altera devices.

Question 3: Does the compiler use optimization techniques to produce JEDEC code ? (i.e. logic minimization)
Answer: Yes, logic minimization, DeMorgan's inversion of outputs, and automatic fitting of resources and pins to the given device are supported in an integrated fashion.

Question 4: Are there alternate logic entry tools, like schematic capture and Finte State Machine entry, are there hooks for simulation and other design methodologies, like gate arrays ?
Answer: Alternate enrty methods include schematic capture, FSM entry, and Graphical Netlist entry(LB). There are currently no tools for functional simulation, nor is ther any way of interfacing to other design tools. Both topics are being considered for the future.

The third party HDLs', like ABEL, CUPL, and LOG/IC are, in general, harder to learn and use. They afford greater expressivness and generality in addressing design problems. Because of the need to work with most devices on the market, these HDL's resemble more closely their high-power cousins in the Custom/Semi-Custom design arena. They have no automatic resource fitting and pin-assignment, but do have a robust set of integrated tools involving functional simulation, schematic capture and FSM entry available.

3. PLD Optimization Tools

Here we introduce and describe the current mix of optimization tools in the PLD development workshop. The goal of CAD optimization tools for PLD design is to speed up the design cycle by reducing designer time, and to cram larger designs into a fixed device. The optimization tools of a typical PLD development

system usually reside in the HDL compiler. The tools can be viewed as compiler optimization tools.

The four optimization tools of concern to us are:

1. **Logic Minimization** - A tool to reduce the complexity of the logic equations implementing a design.

2. **Finite State Machine(FSM) Compiler** - A tool which takes an FSM description and translates or compiles it into an HDL.

3. **DeMorgan's Inversion** - A tool which can reduce the amount ogf logic needed by inverting the sense of some of the output signals of the device.

4. **Fitting and Pin Assignment** - A tool which automatically fits the resources and pins which the user chooses not to.

We discuss each topic seperately, and illustrate, by the use of the Dice Example, (Figures1-3) most of these features.

Logic Minimization:

Logic minimization is currently the best optimization tool available for PLDs'. Logic minimization for current PLD development systems is strictly a two-level logic tool, and totally replaces, for combinational logic purposes, the Karnaugh Map and Quine-McCluskey Algorithm methodology for finding the minimum size set of equations. For two-level logic(AND/OR), most compilers use a single or multiple output heuristic minimizer. University research and industrial experience show that the NP-complete problem of two-level logic minimization has been effectively solved for the size of problems currently being considered in practice. Currently, the most effective minimizers are Espresso, McBoole, and Presto-II. Espresso has been shown, in aggregate, to be within one-percent of the minimum answer on 104 test cases. On current PLD size problems, the test cases indicate that Espresso will find the minimum solution almost always [1, 2; 3, 4].

The user of minimization tools usually has a concern about the process. The minimization tools have the side-effect of producing a reduced equation set that doesn't always reflect the designer's thought process relative to the un-minimized equation set. This results in confusion in trying to understand the design from the reduced equation set. For the same reason, editing a JEDEC file to change a few bits of a design is error-prone and to be avoided. To independently verify the meaning of the design and the resulting minimized boolean equations, functional test vectors should be created and run through a functional simulator. This will catch design errors and give a truth-table verification of the design. For some HDLs', there are tools available that will automatically generate test vectors. Actually, due to the fixed architectures of PLDs', automatic test generation is much more feasible in the PLD arena than in the Custom/Semi-Custom area. Of course, a designer can choose to not have the equations minimized if they already fit into the target device before minimization, but in complicated designs, this is rarely the case, as the Dice Example at the end of this paper shows.

Future PLD tools will depend more and more on logic minimization. Just as a high-level language programmer looks less and less at the object code produced by a compiler, the logic designer will not be concerned with the minimized code output of the PLD logic compiler.

FSM Compiler:

An FSM compiler is included in the list of PLD optimization tools because it allows for a compact representation of a sequential circuit. This leads to ways to introduce systematic optimization techniques like logic minimization and automatic state assignment at a transparent level for the user. Designing at the level of states and transitions results in a more compact HDL description for the same logic function, with fewer mistakes being made. FSM description also allows the user to change memory element types and state assignments in an error-free manner. FSM's are also well understood, theoretically, and are a ripe area for future tool development. Most logic texts present many hand tabular methods for optimizing FSM designs. These hand methods can be automated and extended to produce new tools for FSM-based design.

DeMorgan's Inversion:

DeMorgan's inversion, in AND/OR type PLD architectures, with inversion control in the I/O macrocells, refers to logically inverting an output signal phase in such a way that the number of p-terms realizing the complement function is less than the original function. This can save the user from an un-solvable p-term fitting problem due to too many p-terms when using one sense of an equation. For devices with single output macrocells, like PALs, and EPLDs, thecomplement of the single output equation is computed and then minimized. The sense of the equation with the least number of p-terms is then the one that is implemented in the device under programming.

Fitting and Pin Assignment:

The fitting and pin assignment problem refers to compiling a design file, and having the compiler automatically choose those device resources and pins that the user did not assign in the design file. In the past, device architectures have been simple enough and small enough so that fitting and pin assignment were not a problem for the user. Now, with increasing size, complexity, and non-homogeneity of the device architecture, a heuristic CAD tool, which is like an automatic place and route tool, is a necessity. If a device architecture is homogeneous with respect to structure and resources, fitting is not a problem, as there is no contention for resources or placement of those resources. Fitting is a problem when there are multiple clocks and types of clocks, multiple device sections(like quadrants), varying numbers of p-terms per quadrant, product term sharing and steering, input pins, I/O macrocells of varying types, or buried registers. The greater the number and size of the features, the greater the fitting problem. Without a tool to help, the designer must do the fitting by hand, leading to errors and not finding an allowable fit.

Some of the large scale devices that exhibit these problems are Intel's(Altera's) 5C121(EP1210) and 5C180(EP1800). The fitting and automatic pin assignment tools of iPLDS relieve the user from having to deal with this problem.

3. Future PLD Optimization Tools

This section describes new directions for PLD development systems optimization tools. Optimization tools must be near transparent to the user to get universal acceptance. Of the four optimization tools mentioned above, all but the FSM compiler tool satisfy that criterion.

There are basically two types of optimization tools which will appear in the PLD arena. The first type are tools which are ported from, or interfaced to the Custom/Semi-Custom environment. The current logic description and synthesis tools of silicon compilers and Custom/Semi-Custom CAD tools fit into this classification. The second type are new tools which will address the architecture-specific optimization issues. The tools in this group will use methods based on logic optimization and expert-system techniques. These two methodologies will be applied to taking abstact specifications and realizing them automatically into multiple devices, or in taking multiple abstract specifications and realizing them in one device.

Portation of Custom/Semi-Custom Tools:

Available ideas ready for porting to the PLD environment down the HDL path include implementing a subset of VHDL(VHSIC Hardware Description Language)[5], and having the compiler produce an EDIF(Electronic Design Interchange Format) [6] intermediate format. In this way, interfacing with other toolboxes of any type will be easier. New device support will also be easier, given the generic nature of VHDL. Using VHDL would also standardize an HDL, and allow designers to learn one HDL which will last for a long time. Also, PLD tools which interface with the Custom/Semi-Custom toolset involving board design, testing and manufacturing is needed now, and is being addressed by the major CAD vendors. Standardization, like VHDL and EDIF will, eventually, lower the cost of these interfaces.

The new logic minimization algorithms, like Espresso, and new state assignment tools, like KISS[7] and STASH [8]can be used in the PLD environment. The algorithms and methods of tools involving placement and routing can be applied to the fitting/pin assignment problem. On the logic synthesis side, new tools which combine expert-systems with multi-level logic optimization can be applied to PLD devices which allow multi-level logic to be easily implemented.

The key point, irregardless of the actual tools from the Custom/Semi-Custom arena which are productized is that the user have an essentially transparent view of any new optimization tools.

Once a PLD is manufactured, the functionality cannot be changed. This fact leads to the belief that tools can be created which map logic, which is too big or too slow, into multiple devices by doing automatic logic partitioning. The converse problem of fitting multiple chunks of communicating logic into one device may also be addressed. Tools to fit multiple state machines into one device, or to partition a schematic or FSM into two or more devices is a first step. For example, the Dice Example(Figures 1-3) has three small state machines, which are integrated into one device and design file. Expert-systems can capture the rules for partitioning, and the database of all allowable devices, while optimization techniques can make the expert-systems work as well as, or better than a logic designer.

Conclusion:

We have surveyed the reasons for, and components of PLD development systems, with emphasis on the Hardware Description Languages, and optimization methods in such systems. The conclusions of the survey are that the HDL's are the essential cornerstone of any PLD system, and will control the future directions of any new PLD development tools. The second conclusion is that two-level logic minimization, FSM compiler, and automatic fitting tools are the most important in the PLD optimization area. Also, recent breakthroughs and public availability of heuristic minimizers point to increased use of such tools. Finally, future directions, and an expanding market indicate a wide range of new tools will appear. The key emphasis will be on making them transparent to the user, who, when all is said and done, knows how to design logic best !

References

[1] R. Rudell and A. Sangiovanni-Vincentelli, "ESPRESSO-MV: Algorithms for Multiple Valued Logic Minimization", in Proc. Cust. Int. Circ. Conf., IEEE, Portland, OR, May, 1985.

[2] M.R. Dagenais, V.K. Agarwal and N.C. Rumin, "McBoole: A New Procedure for Exact Logic Minimization", IEEE Trans. on CAD, Jan. 1986, 229-238.

[3] M. Bartholomeus and H.D. Man, "Presto-II: Yet Another Logic Minimizer for Programmed Logic Arrays", Proc. Int. Symp. Circ. Syst., June 1985, 58.

[4] R. Rudell, "Multiple-Valued Logic Minimization for PLA Synthesis", M.S. Thesis, University of California, Berkeley, 1986.

[5] V. D. Agrawal, ed., "VHDL: The VHSIC Hardware Description Language", IEEE Design and Test of Computers, April, 1986.

[6] J.P. Eurich, "A Tutorial Introduction to the Electronic Design Interchange Format", In Proc. of 23rd Design Automation Conference, July, 1986, 327-333.

[7] G. DeMicheli, R.K. Brayton, and A. Sangiovanni-Vincentelli, "Optimal State Assignment for Finite-State Machines", IEEE Trans. on CAD, July, 1985, 269-285.

[8] A. J. Coppola, "An Implementation of a State Assignment Heuristic", In Proc. of 23rd Design Automation Conference, July, 1986, 643-649.

Dice Example Description

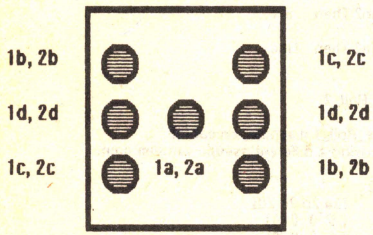
Problem: Design a circuit that will roll two dice. Push a switch to start the dice rolling. When the switch is released, a (pseudo) random set of numbers will be displayed.

The example is written using the FSM compiler module of iPLDS. This example is a modification of an existing Application Note[AP-279] design, which is written in ADF language.

The Dice Example pseudo-randomly rolls two dice. The Dice Example is composed of three FSMs'. The first two are essentially up-counters, which count from one to six, using the notation groups of one or two LED's, for each of four outputs, to represent the six faces of a die. A picture which indicates the LED groupings, by listing the output signal name next to the LED controlled by it, is given in Diagram 1. The groupings for both die are identical, and hence, listed next to each other.

The third machine generates a short pseudo-random bit sequnce by implementing a Linear Feedback Shift Register(LFSR) , with three registers. The pseudo-random bit sequences from the LFSR are used to add probabilistic transitions to the up-counter model of each die. The implementation of the LFSR is by straight memorization of the sequence, by means of the state variables of an up-counter.

Dice LED Encoding



Die 1 Signals: 1a, 1b, 1c, 1d
Die 2 Signals: 2a, 2b, 2c, 2d

The Dice Example shows the usefulness of logic minimization. Figure 2 shows the FSM Language(State Machine File) representation of the Dice Example under the iPLDS system. Figure 3 shows the ADF code which resulted, as an intermediate step, in the compilation process.

The target device, the 5C060, has 16 I/O macrocells, but each macrocell has only enough room for 8 p-terms. There are 11 equations that result from the Dice Example design. Four for each die, to control the LED's, and three from the state variables of the Linear Feedback Shift Register.

We present a before and after minimization table, showing the effect of the minimization and the automatic DeMorgan's Inversion step.

Equation	Inputs	p-terms before min	p-terms after min
Sv3.d	3	3	2
Sv2.d	3	3	2
Sv1.d	3	3	2
2d.d	6	3	3
2o.d	6	9	4
2b.d	6	6	4
2a.d	6	10	6
1d.d	6	3	3
1c.d	6	9	4
1b.d	6	5	4
1a.d	6	10	7

A necessary condition to fit into the 5C060 is that all of the numbers in the last column be no more than 8, as there are no more than 8 p-terms connected to any macrocell. This particular problem took 2 minutes of CPU time on an 8Mhz PC/AT. Reducing these equations by hand, even for this simple example, would be difficult. The need for the automatic minimizer is clear in this example. Without it, a designer would either have to reduce the eqations resulting from the FSM Language by hand, or not use an FSM Language at all, and do the whole design using hand-crafted methods.

Figure 1.

Dice Example
FSM Language(SMF) Description

```
Alan Coppola
Intel
July 21, 1986
Part No.: LasVegas
Ver. 3.0
5C060
Roll a pair of die
LB Version 4.01, Baseline 27.1 4/9/86
PART: 5C060

% No pins assigned:
  Automatic Pin Assignment and Fitting %

INPUTS: clk1, clk2, Go
OUTPUTS: 1a, 1b, 1c, 1d, 2a, 2b, 2c, 2d

NETWORK:
   clk1 = INP(clk1)
   clk2 = INP(clk2)
    Go  = INP(Go)
%
Three term LFSR, implemented by storing sequence
in state variables, which act as flipping coins.
%

MACHINE: LFSR
CLOCK: clk2

STATES: [Coin1 Coin2 Coin3]
S0     [0 0 0]
S1     [1 0 0]
S2     [1 1 0]
S3     [0 1 1]
S4     [1 0 1]
S5     [0 1 0]
S6     [0 0 1]
%
State equations are:
   Coin2 := Coin1
   Coin3 := Coin2
   Coin1 := /(Coin2 xor Coin3)
%

S0:
   S1
S1:
   S2
S2:
   S3
S3:
   S4
S4:
   S5
S5:
   S6
S6:
   S0

MACHINE: Die_Roll_1
   CLOCK: clk1

%
State variables are used as outputs to die.
Each state encodes the set of LED's to light
to realize that die value.
%
   STATES: [1a 1b 1c 1d]
   Reset        [ 0  0  0  0]
   One          [ 1  0  0  0]
   Two          [ 0  1  0  0]
   Three        [ 1  1  0  0]
   Four         [ 0  1  1  0]
   Five         [ 1  1  1  0]
   Six          [ 0  1  1  1]

%
Coin2 is a pseudo-random coin, which controls the
up-counter transitions, so that the dice roll is
pseudo-random.
%
   Reset:
      If Go  Then  One
   One:
      If Go*Coin2 Then   Two
   Two:
      If Go*Coin2 Then   Three
   Three:
      If Go*Coin2 Then    Four
   Four:
      If Go*Coin2 Then    Five
   Five:
      If Go*Coin2 Then    Six
   Six:
      If Go*Coin2 Then   One


MACHINE: Die_Roll_2
%
Duplicate of Die_Roll_1 machine, except for
a different die, using a different pseudo-random coin.
%
   CLOCK: clk2
   STATES:      [2a 2b 2c 2d]
   ResetDie2    [ 0  0  0  0]
   OneDie2      [ 1  0  0  0]
   TwoDie2      [ 0  1  0  0]
   ThreeDie2    [ 1  1  0  0]
   FourDie2     [ 0  1  1  0]
   FiveDie2     [ 1  1  1  0]
   SixDie2      [ 0  1  1  1]

   ResetDie2:
      If Go*Coin3 Then   OneDie2
   OneDie2:
      If Go*Coin3 Then   TwoDie2
   TwoDie2:
      If Go*Coin3 Then   ThreeDie2
   ThreeDie2:
      If Go*Coin3 Then   FourDie2
   FourDie2:
      If Go*Coin3 Then   FiveDie2
   FiveDie2:
      If Go*Coin3 Then   SixDie2
   SixDie2:
      If Go*Coin3 Then   OneDie2


END$
```

Figure 2.

Dice Example
Hardware Description Language(ADF)

```
Alan Coppola
Intel
July 21, 1986
Part No.: LasVegas
Ver. 3.0
5C060
Roll a pair of die
LB Version 4.01, Baseline 27.1 4/9/86
SMV Version 1.01 BETA2 Baseline 26.1 4/3/86
PART: 5C060

INPUTS:
clk1, clk2, Go

OUTPUTS:
1a, 1b, 1c, 1d, 2a, 2b, 2c, 2d

NETWORK:

   clk1 = INP(clk1)
   clk2 = INP(clk2)
   Go  = INP(Go)
%
Three term LFSR, implemented by storing sequence
in state variables, which act as flipping coins.
%

%
I/O's for State Machine "LFSR"
%
Coin1 = NORF(Coin1.d, clk2, GND, GND)
Coin2 = NORF(Coin2.d, clk2, GND, GND)
Coin3 = NORF(Coin3.d, clk2, GND, GND)
%
I/O's for State Machine "Die_Roll_1"
%
1a, 1a = RORF(1a.d, clk1, GND, GND, VCC)
1b, 1b = RORF(1b.d, clk1, GND, GND, VCC)
1c, 1c = RORF(1c.d, clk1, GND, GND, VCC)
1d, 1d = RORF(1d.d, clk1, GND, GND, VCC)
%
I/O's for State Machine "Die_Roll_2"
%
2a, 2a = RORF(2a.d, clk2, GND, GND, VCC)
2b, 2b = RORF(2b.d, clk2, GND, GND, VCC)
2c, 2c = RORF(2c.d, clk2, GND, GND, VCC)
2d, 2d = RORF(2d.d, clk2, GND, GND, VCC)

EQUATIONS:
%
Boolean Equations for State Machine "LFSR"
%
%
Current State Equations for "LFSR"
%
S0 = Coin1'*Coin2'*Coin3';
S1 = Coin1*Coin2'*Coin3';
S2 = Coin1*Coin2*Coin3';
S3 = Coin1'*Coin2*Coin3;
S4 = Coin1*Coin2'*Coin3;
S5 = Coin1'*Coin2*Coin3';
S6 = Coin1'*Coin2'*Coin3;
%
SV Defining Equations for State Machine "LFSR"
%
Coin1.d = S1.n +  S2.n  + S4.n;
Coin2.d = S2.n +  S3.n  + S5.n;
Coin3.d = S3.n +  S4.n  + S6.n;
%
Next State Equations for State Machine "LFSR"
%
S1.n = S0;
S2.n = S1;
S3.n = S2;
S4.n = S3;
S5.n = S4;
S6.n = S5;
%
Boolean Equations for State Machine "Die_Roll_1"
%
%
Current State Equations for "Die_Roll_1"
%
Reset = 1a'*1b'*1c'*1d';
One = 1a*1b'*1c'*1d';
Two = 1a'*1b*1c'*1d';
Three = 1a*1b*1c'*1d';
Four = 1a'*1b'*1c*1d';
Five = 1a*1b'*1c*1d';
Six = 1a'*1b*1c*1d;
%
SV Defining Equations for State Machine "Die_Roll_1"
%
1a.d = One.n + Three.n  + Five.n;
1b.d' = One.n + Reset.n;
1c.d = Four.n + Five.n + Six.n;
1d.d = Six.n;
%
Next State Equations for State Machine "Die_Roll_1"
%
One.n = Six * Go * Coin2 + One * (Go * Coin2)'
     + Reset * Go;
Reset.n = Reset * (Go)';
Three.n = Three * (Go * Coin2)' + Two * Go * Coin2;
Four.n = Four * (Go * Coin2)'     + Three * Go * Coin2;
Five.n = Five * (Go * Coin2)'     + Four * Go * Coin2;
Six.n = Six * (Go * Coin2)'       + Five * Go * Coin2;
%
Boolean Equations for State Machine "Die_Roll_2"
%
%
Current State Equations for "Die_Roll_2"
%
ResetDie2 = 2a'*2b'*2c'*2d';
OneDie2 = 2a*2b'*2c'*2d';
TwoDie2 = 2a'*2b*2c'*2d';
ThreeDie2 = 2a*2b*2c'*2d';
FourDie2 = 2a'*2b*2c*2d';
FiveDie2 = 2a*2b*2c*2d';
SixDie2 = 2a'*2b'*2c*2d;
%
SV Defining Equations for State Machine "Die_Roll_2"
%
2a.d = OneDie2.n + ThreeDie2.n  + FiveDie2.n;
2b.d' = OneDie2.n + ResetDie2.n;
2c.d = FourDie2.n  + FiveDie2.n  + SixDie2.n;
2d.d = SixDie2.n;
%
Next State Equations for State Machine "Die_Roll_2"
%
OneDie2.n = SixDie2 * Go * Coin3
       + OneDie2 * (Go * Coin3)'
       + ResetDie2 * Go * Coin3;
ResetDie2.n = ResetDie2 * (Go * Coin3)';
ThreeDie2.n = ThreeDie2 * (Go * Coin3)'
        + TwoDie2 * Go * Coin3;
FourDie2.n = FourDie2 * (Go * Coin3)'
      + ThreeDie2 * Go * Coin3;
FiveDie2.n = FiveDie2 * (Go * Coin3)'
       + FourDie2 * Go * Coin3;
SixDie2.n = SixDie2 * (Go * Coin3)'
       + FiveDie2 * Go * Coin3;

END$
```

Figure 3.

Appendix 5

APPENDIX

SECOND SOURCE CROSS REFERENCE

What Intel Part to Quote	Altera Part #
D5C031-50	EP310DC
D5C031-35	EP310DC-2
TD5C031-50	EP310D1
D5C032-35	EP320DC
D5C032-35	EP320DC-2
D5C032-25	EP320DC-1
P5C032-35	EP320PC
P5C032-35	EP320PC-2
D5C060-55	EP600DC
D5C060-35	EP600DC-2
D5C060-45	EP600DC-3
CJ5C060-55	EP600JC
CJ5C060-45	EP600JC-3
P5C060-55	EP600PC
P5C060-45	EP600PC-3
N5C060-55	EP600LC
N5C060-45	EP600LC-3
TD5C060-55	EP600DI
TCJ5C060-55	EP600JI
MD5C060-55 Spec	EP600DM
MD5C060-55	EP600DMB
D5C090-60	EP900DC
D5C090-45	EP900DC-2
D5C090-50	EP900DC-3
CJ5C090-60	EP900JC

What Intel Part to Quote	Altera Part #
CJ5C090-50	EP900JC-3
P5C090-60	EP900PC
P5C090-50	EP900PC-3
N5C090-60	EP900LC
N5C090-50	EP900LC-3
TD5C090-60	EP900DI
TCJ5C090-60	EP900JI
D5C121-90	EP1210DC
D5C121-65	EP1210DC-2
CJ5C121-90	EP1210JC
CJ5C121-65	EP1210JC-2
P5C121-90	EP1210PC
P5C121-65	EP1210PC-2
N5C121-90	EP1210LC
N5C121-65	EP1210LC-2
TD5C121-90	EP1210DI
TCJ5C121-90	EP1210JI
CJ5C180-90	EP1800JC
CJ5C180-75	EP1800JC-3
N5C180-90	EP1800LC
N5C180-75	EP1800LC-3
TCJ5C180-90	EP1800JI
In Development	EP1800GC
In Development	EP1800GCM
In Development	EP1800JM
In Development	EP1800JMB

PLA TO EPLD REPLACEMENT

Already in wide use throughout the electronics industry are numerous different Programmable Logic Devices. Many of these are PALs from MMI. Currently, two of our EPLD products, the 5C060 and 5C031 can functionally replace most 24-pin and 20-pin PALs, respectively. A third product, the 5AC312, with its architecturally advanced features, can replace most designs using more complex PALs such as the 20RA10, 22Y10, and 32V10.

The 5C031

The 5C031 is a direct, drop-in replacement for most 20-pin PALs, although some PALs have an incompatible architecture.

The 5C060

The 5C060 is NOT a drop-in replacement for any 24-pin PAL, though it can functionally replace most. The reason for this is that pin 1 is used as the main clock on registered PALs and as an input on non-registered. Also, pin 13 is used as an $\overline{OE}$ line on some PALs, and as an input on others. The 5C060, however, uses pin 1 as the left-half synchronous clock input and pin 13 as the right-half synchronous clock input.

While that may not be a problem in some PAL designs, those designs that require clocking or inputs on pins 1 or 13 will necessitate hardware modifications. In the case of the registered PALs, the connection to pin 1 must be rerouted to pin 13 and the OE connected to one of the available inputs (if used). In this manner, the 5C060 can functionally replace the PAL.

The 5AC312

The 5AC312 is a direct, drop-in replacement for the 20RA10 as well as many of the other simple 24-pin logic devices. The 5AC312 can also serve as a drop-in replacement for most designs using the 22V10 or 32V10 devices.

5C031/5C032 As a 20-Pin PAL Replacement

100% Compatible	Functionally Compatible
10H8, -2	
12H6, -2	
14H4, -2	
16H2, -2	
10L8, -2	
12L6, -2	16R6A
16L8, A-2, A-4	16R4A
16R4, A-2, A-4	16L8A
14L4, -2	16RP6A
16L2, -2	16RP4A
16R8, A-2, A-4	16P8A
16R6, A-2, A-4	16R8A
16P8, -2	16RP8A
16RP8, -2	
16RP6, -2	
16RP4, -2	
These are 25 ns–45 ns PALs.	These are 15 ns PALs.

5C060 As a 24-Pin PAL Replacement

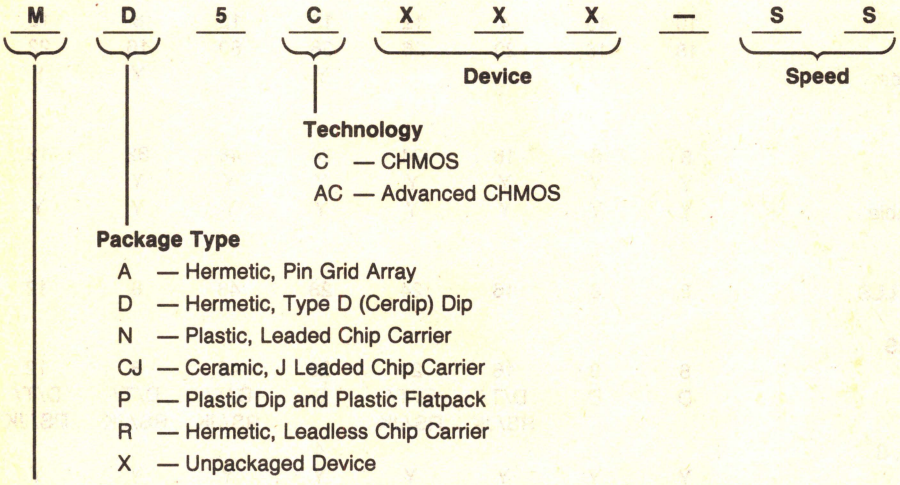
Modified Replacement	Functionally Compatible
12L10	20L8A
14L8	20R8A
16L6	20R6A
18L4	20R4A
20L2	
20L10	
20L8	
20R8	
20R6	
20R4	
20RA10	
With hardware modifications	These are 15 ns PALs.

5AC312 As a 24-Pin PAL Replacement

100% Compatible	100% Compatible (Qualified)
20L8	22V10
20R8	32V10
20R6	Dependent on the
20R4	number of product
20RA10	terms used.

ORDERING INFORMATION

Intel EPLDs are identified as follows:


M D 5 C X X X — S S
Device
Speed
Technology
C — CHMOS
AC — Advanced CHMOS
Package Type
A — Hermetic, Pin Grid Array
D — Hermetic, Type D (Cerdip) Dip
N — Plastic, Leaded Chip Carrier
CJ — Ceramic, J Leaded Chip Carrier
P — Plastic Dip and Plastic Flatpack
R — Hermetic, Leadless Chip Carrier
X — Unpackaged Device

A — Indicates automotive operating temperature range (−40°C to +125°C)

J — Indicates a JAN qualified device, but is for internal identification purposes only. All JAN devices must be ordered by M38510 part number. (Example: M38510/42001 BQB), and will be marked in accordance with MIL-M-38510 specifications.

L — Indicates extended operating temperature range (−40°C to +85°C) express product with 160 ±8 hrs. dynamic burn-in.

M — Indicates military operating temperature range (−55°C to +125°C)

Q — Indicates commercial temperature range (0°C to 70°C) express product with 160 ±8 hrs. dynamic burn-in.

T — Indicates extended temperature range (−40°C to +85°C) express product without burn-in.

— No letter indicates commercial temperature range (0°C to 70°C) without burn-in.

Examples:

QD5C060-45 Commercial with burn-in, ceramic Dip, 060 (600 gate) device, 45 nanosecond.

*On military temperature devices, B suffix indicates MIL-STD-883C level B processing.

Device Feature Comparison

	5C031	5C032	5C060	5C090	5C121	5C180	5CBIC	5AC312
INPUTS								
Dedicated	10	10	4	12	12	12	8	10
Maximum	18	18	20	36	36	60	16	22
Input Latches					Y		Y	Y
I/O								
Number	8	8	16	24	24	48	32	12
Tri-State	Y	Y	Y	Y	Y	Y	Y	Y
Programmable Polarity	Y	Y	Y	Y	Y	Y	Y	Y
MACROCELLS	8	8	16	24	28	48	8	12
REGISTERS								
Number	8	8	16	24	28	48	8	12
Types	D	D	D/T/ RS/JK	D/T/ RS/JK	D	D/T/ RS/JK	D/T/ RS/JK	D/T/ RS/JK
Buried Reg. S					4			
Preload	Y	Y	Y	Y	Y	Y	Y	Y
By-Pass	Y	Y	Y	Y	Y	Y	Y	Y
Reset	Y	Y	Y	Y	Y	Y	Y	Y
Preset	Y						Y	Y
PRODUCT TERMS								
Number	74	72	160	240	236	480	112	200
Sharing					Y			
Variable Prod. Term Distribution					Y			
LOCAL/GLOBAL BUSSES					Y	Y		
CLOCKS	1	1	2	2	2	4		2
Asynchronous Clocking			Y	Y		Y	Y	Y
Programmable Clock Edges					Y			
SECURITY BIT	Y	Y	Y	Y	Y	Y	Y	Y

ELPD CUSTOMER SUPPORT

Hotline

The Intel EPLD Technical Hotline is manned by application personnel from 8:00 a.m. to 5:00 p.m. (PST) every business day. Contact your local field sales office for the hotline number.

BBS

Intel has a Bulletin Board System for registered iPLS and iPLS II customers to electronically transfer information. Any registered person with a modem can log onto the system. The current number is (916) 985-2308. If your communication software supports file transfers, you can receive utilities, software updates, and the latest information on EPLDs via the Bulletin Board.

EPLD Customer Design Support Center

Intel has a Customer Design Support Center to help customers who are implementing EPLD designs. Service includes answering questions, device selection assistance, and design partitioning as well as limited prototyping, and product/design evaluation and implementation. For more information on the Design Support Center, contact your local Intel field sales office.

EPLD Evaluation Unit

A modular unit for evaluation of EPLD devices is available from Intel. The unit has a variety of switches and LEDs, and a numeric display for control and status. Several Intel applications can be verified on the unit as well as small customer designs. For more information, contact your local Intel field sales office.

COMPATIBLE COMPUTERS FOR iPLDS II

A partial list of computers that have been verified to be software compatible with the Intel Programmable Logic Development System (iPLDS II) is given below:

AT&T 6300 and 6300+
Compaq family of PCs (88, 86, 286, 386)
IBM AT
IBM XT
IBM XT-286
HP Vectra
Sperry IT
Tandy 3000 HD

DOMESTIC SALES OFFICES

ALABAMA

†Intel Corp.
5015 Bradford Dr., #2
Huntsville 35805
Tel: (205) 830-4010

ARIZONA

†Intel Corp.
11225 N. 28th Dr., #D214
Phoenix 85029
Tel: (602) 869-4980

Intel Corp.
1161 N. El Dorado Place
Suite 301
Tucson 85715
Tel: (602) 299-6815

CALIFORNIA

†Intel Corp.
21515 Vanowen Street
Suite 116
Canoga Park 91303
Tel: (818) 704-8500

†Intel Corp.
2250 E. Imperial Highway
Suite 218
El Segundo 90245
Tel: (213) 640-6040

Intel Corp.
1510 Arden Way, Suite 101
Sacramento 95815
Tel: (916) 920-8096

†Intel Corp.
4350 Executive Drive
Suite 105
San Diego 92121
Tel: (619) 452-5880

Intel Corp.*
400 N. Tustin Avenue
Suite 450
Santa Ana 92705
Tel: (714) 835-9642
TWX: 910-595-1114

†Intel Corp.*
San Tomas 4
2700 San Tomas Expressway
Santa Clara, CA 95051
Tel: (408) 986-8086
TWX: 910-338-0255

COLORADO

Intel Corp.
4445 Northpark Drive
Suite 100
Colorado Springs 80907
Tel: (303) 594-6622

†Intel Corp.*
650 S. Cherry St., Suite 915
Denver 80222
Tel: (303) 321-8086
TWX: 910-931-2289

CONNECTICUT

†Intel Corp.
26 Mill Plain Road
Danbury 06811
Tel: (203) 748-3130
TWX: 710-456-1199

FLORIDA

†Intel Corp.
242 N. Westmonte Dr.
Suite 105
Altamonte Springs 32714
Tel: (305) 869-5588
FAX: 305-682-6047

Intel Corp.
6363 N.W. 6th Way, Suite 100
Ft. Lauderdale 33309
Tel: (305) 771-0600
TWX: 510-956-9407
FAX: 305-772-8193

Intel Corp.
11300 4th Street North
Suite 170
St. Petersburg 33702
Tel: (813) 577-2413
FAX: 813-578-1607

GEORGIA

†Intel Corp.
3280 Pointe Parkway
Suite 200
Norcross 30092
Tel: (404) 449-0541

ILLINOIS

Intel Corp.*
300 N. Martingale Road, Suite 400
Schaumburg 60173
Tel: (312) 310-8031

INDIANA

†Intel Corp.
8777 Purdue Road
Suite 125
Indianapolis 46268
Tel: (317) 875-0623

IOWA

Intel Corp.
St. Andrews Building
1930 St. Andrews Drive N.E.
Cedar Rapids 52402
Tel: (319) 393-5510

KANSAS

†Intel Corp.
8400 W. 110th Street
Suite 170
Overland Park 66210
Tel: (913) 345-2727

MARYLAND

Intel Corp.*
7321 Parkway Drive South
Suite C
Hanover 21076
Tel: (301) 796-7500
TWX: 710-862-1944

Intel Corp.
5th Floor
7833 Walker Drive
Greenbelt 20770
Tel: (301) 441-1020

MASSACHUSETTS

†Intel Corp.*
Westford Corp. Center
3 Carlisle Road
Westford 01886
Tel: (617) 692-3222
TWX: 710-343-6333

MICHIGAN

†Intel Corp.
7071 Orchard Lake Road
Suite 100
West Bloomfield 48033
Tel: (313) 851-8096

MINNESOTA

Intel Corp.
3500 W. 80th St., Suite 360
Bloomington 55431
Tel: (612) 835-6722
TWX: 910-576-2867

MISSOURI

Intel Corp.
4203 Earth City Expressway
Suite 131
Earth City 63045
Tel: (314) 291-1990

NEW JERSEY

Intel Corp.*
Parkway 109 Office Center
328 Newman Springs Road
Red Bank 07701
Tel: (201) 747-2233

Intel Corp.
280 Corporate Center
75 Livingston Avenue
First Floor
Roseland 07068
Tel: (201) 740-0111

NEW MEXICO

Intel Corp.
8500 Menual Boulevard N.E.
Suite B 295
Albuquerque 87112
Tel: (505) 292-8086

NEW YORK

Intel Corp.
127 Main Street
Binghamton 13905
Tel: (607) 773-0337

Intel Corp.*
850 Cross Keys Office Park
Fairport 14450
Tel: (716) 425-2750
TWX: 510-253-7391

Intel Corp.*
300 Motor Parkway
Hauppauge 11787
Tel: (516) 231-3300
TWX: 510-227-6236

Intel Corp.
Suite 2B Hollowbrook Park
15 Myers Corners Road
Wappinger Falls 12590
Tel: (914) 297-5161
TWX: 510-248-0060

NORTH CAROLINA

Intel Corp.
5700 Executive Center Drive
Suite 213
Charlotte 28212
Tel: (704) 568-8966

†Intel Corp.
2700 Wycliff Road
Suite 102
Raleigh 27607
Tel: (919) 781-8022

OHIO

Intel Corp.*
3401 Park Center Drive
Suite 220
Dayton 45414
Tel: (513) 890-5350
TWX: 810-450-2528

Intel Corp.*
25700 Science Park Dr., Suite 100
Beachwood 44122
Tel: (216) 464-2736
TWX: 810-427-9298

OKLAHOMA

Intel Corp.
6801 N. Broadway
Suite 115
Oklahoma City 73116
Tel: (405) 848-8086

OREGON

†Intel Corp.
15254 N.W. Greenbrier Parkway, Bldg. B
Beaverton 97006
Tel: (503) 645-8051
TWX: 910-467-8741

PENNSYLVANIA

Intel Corp.
1513 Cedar Cliff Drive
Camp Hill 17011
Tel: (717) 737-5035

Intel Corp.*
455 Pennsylvania Avenue
Fort Washington 19034
Tel: (215) 641-1000
TWX: 510-661-2077

Intel Corp.*
400 Penn Center Blvd., Suite 610
Pittsburgh 15235
Tel: (412) 823-4970

PUERTO RICO

Intel Microprocessor Corp.
South Industrial Park
P.O. Box 910
Las Piedras 00671
Tel: (809) 733-8616

TEXAS

†Intel Corp.
313 E. Anderson Lane
Suite 314
Austin 78752
Tel: (512) 454-3628

†Intel Corp.*
12300 Ford Road
Suite 380
Dallas 75234
Tel: (214) 241-8087
TWX: 910-860-5617

Intel Corp.*
7322 S.W. Freeway
Suite 1490
Houston 77074
Tel: (713) 988-8086
TWX: 910-881-2490

UTAH

Intel Corp.
5201 Green Street
Suite 290
Murray 84123
Tel: (801) 263-8051

VIRGINIA

†Intel Corp.
1504 Santa Rosa Road
Suite 108
Richmond 23288
Tel: (804) 282-5668

WASHINGTON

Intel Corp.
155-108 Avenue N.E.
Suite 386
Bellevue 98004
Tel: (206) 453-8086
TWX: 910-443-3002

Intel Corp.
408 N. Mullan Road
Suite 102
Spokane 99206
Tel: (509) 928-8086

WISCONSIN

†Intel Corp.
330 S. Executive Dr.
Suite 102
Brookfield 53005
Tel: (414) 784-8087
FAX: (414) 796-2115

CANADA

BRITISH COLUMBIA

Intel Semiconductor of Canada, Ltd.
4585 Canada Way, Suite 202
Burnaby V5G 4L6
Tel: (604) 298-0387
FAX: (604) 298-8234

ONTARIO

†Intel Semiconductor of Canada, Ltd.
2650 Queensview Drive
Suite 250
Ottawa K2B 8H6
Tel: (613) 829-9714
TLX: 053-4115

†Intel Semiconductor of Canada, Ltd.
190 Attwell Drive
Suite 500
Rexdale M9W 6H8
Tel: (416) 675-2105
TLX: 06983574
FAX: (416) 675-2438

QUEBEC

†Intel Semiconductor of Canada, Ltd.
620 St. Jean Boulevard
Pointe Claire H9R 3K3
Tel: (514) 694-9130
TWX: 514-694-9134

†Sales and Service Office
*Field Application Location

DOMESTIC DISTRIBUTORS

ALABAMA

Arrow Electronics, Inc.
1015 Henderson Road
Huntsville 35816
Tel: (205) 837-6955

†Hamilton/Avnet Electronics
4940 Research Drive
Huntsville 35805
Tel: (205) 837-7210
TWX: 810-726-2162

Pioneer/Technologies Group Inc.
4825 University Square
Huntsville 35816
Tel: (205) 837-9300
TWX: 810-726-2197

ARIZONA

†Hamilton/Avnet Electronics
505 S. Madison Drive
Tempe 85281
Tel: (602) 968-1461
TWX: 910-950-0077

Kierulff Electronics, Inc.
4134 E. Wood Street
Phoenix 85040
Tel: (602) 437-0750
FAX: 602-252-9109

Wyle Distribution Group
17855 N. Black Canyon Highway
Phoenix 85023
Tel: (602) 866-2888
FAX: 602-866-6937

CALIFORNIA

Arrow Electronics, Inc.
19748 Dearborn Street
Chatsworth 91311
Tel: (818) 701-7500
FAX: 818-772-8930

Arrow Electronics, Inc.
9511 Ridgehaven Court
San Diego 92123
Tel: (619) 565-4800
FAX: 619-279-0862

†Arrow Electronics, Inc.
521 Weddell Drive
Sunnyvale 94089
Tel: (408) 745-6600
FAX: 408-743-4770

Arrow Electronics, Inc.
2961 Dow Avenue
Tustin 92680
Tel: (714) 838-5422
FAX: 714-838-4151

†Avnet Electronics
350 McCormick Avenue
Costa Mesa 92626
Tel: (714) 754-6051
FAX: 714-754-6007

Hamilton/Avnet Electronics
1175 Bordeaux Drive
Sunnyvale 94089
Tel: (408) 743-3300
FAX: 408-745-6679

†Hamilton/Avnet Electronics
4545 Viewridge Avenue
San Diego 92123
Tel: (619) 571-7500
FAX: 619-277-6136

†Hamilton/Avnet Electronics
9650 Desoto Ave.
Chatsworth 91311
Tel: (818) 700-1222, 6500
FAX: 818-700-6553

†Hamilton/Avnet Electronics
4103 Northgate Boulevard
Sacramento 95834
Tel: (916) 920-3150
FAX: 916-925-3478

†Hamilton/Avnet Electronics
3002 G Street
Ontario 91311
Tel: (714) 989-9411
FAX: 714-980-7129

†Hamilton/Avnet Electronics
10950 W. Washington Blvd.
Culver City 90230
Tel: (213) 558-2458
FAX: 213-558-2248

†Hamilton Electro Sales
3170 Pullman Street
Costa Mesa 92626
Tel: (714) 641-4150
FAX: 714-641-4122

CALIFORNIA (Cont'd.)

Kierulff Electronics, Inc.
10824 Hope Street
Cypress 90430
Tel: (714) 220-6300
FAX: 714-821-8420

†Kierulff Electronics, Inc.
1180 Murphy Avenue
San Jose 95131
Tel: (408) 971-2600
FAX: 408-947-3432

†Kierulff Electronics, Inc.
14242 Chamber Rd.
Tustin 92680
Tel: (714) 731-5711
FAX: 714-669-4235

†Kierulff Electronics, Inc.
9800 Variel St.
Chatsworth 91311
Tel: (213) 725-0325
FAX: 818-407-0803

Wyle Distribution Group
26677 W. Agoura Rd.
Calabasas 91302
Tel: (818) 880-9000
FAX: 818-880-5510

†Wyle Distribution Group
17872 Cowan Avenue
Irvine 92714
Tel: (714) 863-9953
FAX: 714-863-0473

Wyle Distribution Group
11151 Sun Center Drive
Rancho Cordova 95670
Tel: (916) 638-5282
FAX: 916-638-1491

†Wyle Distribution Group
9525 Chesapeake Drive
San Diego 92123
Tel: (619) 565-9171
TWX: 910-371-9592
FAX: 619-565-9171 ext. 274

†Wyle Distribution Group
3000 Bowers Avenue
Santa Clara 95051
Tel: (408) 727-2500
FAX: 408-727-5896

Wyle Military
18910 Teller Avenue
Irvine 92715
Tel: (714) 851-9958
TWX: 310-371-9127
FAX: 714-851-8366

Wyle Systems
7382 Lampson Avenue
Garden Grove 92641
Tel: (714) 891-1717
FAX: 714-895-9038

COLORADO

Arrow Electronics, Inc.
1390 S. Potomac Street
Suite 136
Aurora 80012
Tel: (303) 696-1111

†Hamilton/Avnet Electronics
8765 E. Orchard Road
Suite 708
Englewood 80111
Tel: (303) 740-1017
TWX: 910-935-0787

†Wyle Distribution Group
451 E. 124th Avenue
Thornton 80241
Tel: (303) 457-9953
TWX: 910-936-0770

CONNECTICUT

†Arrow Electronics, Inc.
12 Beaumont Road
Wallingford 06492
Tel: (203) 265-7741
TWX: 710-476-0162

Hamilton/Avnet Electronics
Commerce Industrial Park
Commerce Drive
Danbury 06810
Tel: (203) 797-2800
FAX: 203-797-2866

†Pioneer Northeast Electronics
112 Main Street
Norwalk 06851
Tel: (203) 853-1515
TWX: 710-468-3373

FLORIDA

†Arrow Electronics, Inc.
350 Fairway Drive
Deerfield Beach 33441
Tel: (305) 429-8200
TWX: 510-955-9456

Arrow Electronics, Inc.
1001 N.W. 62nd St., Ste. 108
Ft. Lauderdale 33309
Tel: (305) 475-4297
TWX: 510-955-9456

†Arrow Electronics, Inc.
1530 Bottlebrush N.E.
Palm Bay 32905
Tel: (305) 725-1480

†Hamilton/Avnet Electronics
6801 N.W. 15th Way
Ft. Lauderdale 33309
Tel: (305) 971-2900
TLX: 510-956-3097

Hamilton/Avnet Electronics
3245 Tech Drive North
St. Petersburg 33702
Tel: (813) 576-3930
TWX: 810-863-0374

Hamilton/Avnet Electronics
6947 University Boulevard
Winterpark 32792
Tel: (305) 628-3888
FAX: 305-628-3888 ext. 40

†Pioneer Electronics
337 N. Lake Blvd., Ste. 1000
Alta Monte Springs 32701
Tel: (305) 834-9090
TWX: 810-853-0284

Pioneer Electronics
674 S. Military Trail
Deerfield Beach 33442
Tel: (305) 428-8877
TWX: 510-955-9653

GEORGIA

†Arrow Electronics, Inc.
3155 Northwoods Parkway
Suite A
Norcross 30071
Tel: (404) 449-8252
FAX: 404-242-6827

Hamilton/Avnet Electronics
5825 D. Peachtree Corners East
Norcross 30092
Tel: (404) 447-7500
TWX: 810-766-0432

Pioneer Electronics
3100 F. Northwoods Place
Norcross 30071
Tel: (404) 448-1711
FAX: 404-446-8270

ILLINOIS

†Arrow Electronics, Inc.
2000 E. Alonquin Street
Schaumberg 60173
Tel: (312) 397-3440
FAX: 312-397-3550

†Hamilton/Avnet Electronics
1130 Thorndale Avenue
Bensenville 60106
Tel: (312) 860-7780
TWX: 910-227-0060

Kierulff Electronics, Inc.
1140 W. Thorndale
Itasca 60143
Tel: (312) 250-0500
FAX: 312-250-0916

MTI Systems Sales
1100 West Thorndale
Itasca 60143
Tel: (312) 773-2300

†Pioneer Electronics
1551 Carmen Drive
Elk Grove Village 60007
Tel: (312) 437-9680
TWX: 910-222-1834

INDIANA

†Arrow Electronics, Inc.
2495 Directors Row, Suite H
Indianapolis 46241
Tel: (317) 243-9353
TWX: 810-341-3119

INDIANA (Cont'd.)

Hamilton/Avnet Electronics
485 Gradle Drive
Carmel 46032
Tel: (317) 844-9333
FAX: 317-844-5921

†Pioneer Electronics
6408 Castleplace Drive
Indianapolis 46250
Tel: (317) 849-7300
TWX: 810-260-1794

KANSAS

†Hamilton/Avnet Electronics
9219 Quivera Road
Overland Park 66215
Tel: (913) 888-8900
FAX: 913-541-7951

Pioneer Electronics
10551 Lackman Rd.
Lenexa 66215
Tel: (913) 492-0500
FAX: 913-492-7832

KENTUCKY

Hamilton/Avnet Electronics
805-A Newtown Circle
Lexington 40511
Tel: (606) 259-1475
FAX: 606-252-3238

MARYLAND

Arrow Electronics, Inc.
8300 Guilford Road, Ste. H
Rivers Center
Columbia 21046
Tel: (301) 995-6002
TWX: 710-236-9005
FAX: 301-381-3854

†Hamilton/Avnet Electronics
6822 Oak Hall Lane
Columbia 21045
Tel: (301) 995-3500
FAX: 301-995-3593

†Mesa Technology Corp.
9720 Patuxent Woods Dr.
Columbia 21046
Tel: (301) 720-5020
TWX: 710-828-9702

†Pioneer Electronics
9100 Gaither Road
Gaithersburg 20877
Tel: (301) 921-0660
TWX: 710-828-0545

MASSACHUSETTS

†Arrow Electronics, Inc.
1 Arrow Drive
Woburn 01801
Tel: (617) 933-8130
TWX: 710-393-6770

†Hamilton/Avnet Electronics
10D Centennial Drive
Peabody 01960
Tel: (617) 532-3701
TWX: 710-393-0382

Kierulff Electronics, Inc.
13 Fortune Dr.
Billerica 01821
Tel: (617) 667-8331
TWX: 710-390-1449
FAX: 617-663-1754

Pioneer Northeast Electronics
44 Hartwell Avenue
Lexington 02173
Tel: (617) 861-9200
FAX: 617-863-1547

MICHIGAN

Arrow Electronics, Inc.
755 Phoenix Drive
Ann Arbor 48108
Tel: (313) 971-8220
FAX: 313-971-2633

†Hamilton/Avnet Electronics
32487 Schoolcraft Road
Livonia 48150
Tel: (313) 522-4700
TWX: 810-242-8775
FAX: 313-522-2624

Hamilton/Avnet Electronics
2215 29th Street S.E.
Space A5
Grand Rapids 49508
Tel: (616) 243-8805
TWX: 810-273-6921
FAX: 616-243-0028

MICHIGAN (Cont'd.)

Pioneer Electronics
4505 Broadmoor Ave. S.E.
Grand Rapids 49508
Tel: (616) 698-1800
FAX: 616-698-1831

†Pioneer Electronics
13485 Stamford
Livonia 48150
Tel: (313) 525-1800
TWX: 810-242-3271

MINNESOTA

†Arrow Electronics, Inc.
5230 W. 73rd Street
Edina 55435
Tel: (612) 830-1800
FAX: 612-830-1856

Hamilton/Avnet Electronics
12400 White Water Drive
Minnetonka 55343
Tel: (612) 932-0600
FAX: 612-932-0613

†Pioneer Electronics
10203 Bren Road East
Minnetonka 55343
Tel: (612) 935-5444
FAX: 612-935-1921

MISSOURI

†Arrow Electronics, Inc.
2380 Schuetz
St. Louis 63146
Tel: (314) 567-6888
FAX: 314-567-1164

†Hamilton/Avnet Electronics
13743 Shoreline Court East
Earth City 63045
Tel: (314) 344-1200
FAX: 314-291-8889

Kierulff Electronics, Inc.
11804 Borman Dr.
St. Louis 63146
Tel: (314) 997-4956
FAX: 314-567-0860

NEW HAMPSHIRE

†Arrow Electronics, Inc.
3 Perimeter Road
Manchester 03103
Tel: (603) 668-6968
FAX: 603-668-3484

Hamilton/Avnet Electronics
444 E. Industrial Drive
Manchester 03103
Tel: (603) 624-9400
FAX: 603-624-2402

NEW JERSEY

†Arrow Electronics, Inc.
6000 Lincoln Drive East
Marlton 08053
Tel: (609) 596-8000
FAX: 609-596-5632

†Arrow Electronics, Inc.
6 Century Drive
Parsipanny 07054
Tel: (201) 538-0900
FAX: 201-538-4962

†Hamilton/Avnet Electronics
1 Keystone Ave., Bldg. 36
Cherry Hill 08003
Tel: (609) 424-0110
TWX: 710-940-0262
FAX: 609-751-8624

†Hamilton/Avnet Electronics
10 Industrial
Fairfield 07006
Tel: (201) 575-3390
FAX: 201-575-5839

†Pioneer Northeast Electronics
45 Route 46
Pinebrook 07058
Tel: (201) 575-3510
FAX: 201-575-3454

†MTI Systems Sales
37 Kulick Rd.
Fairfield 07006
Tel: (201) 227-5552
FAX: 201-575-6336

†Microcomputer System Technical Distributor Centers

DOMESTIC DISTRIBUTORS

NEW MEXICO

Alliance Electronics Inc.
11030 Cochiti S.E.
Albuquerque 87123
Tel: (505) 292-3360
FAX: 505-292-6537

Hamilton/Avnet Electronics
2524 Baylor Drive S.E.
Albuquerque 87106
Tel: (505) 765-1500
FAX: 505-243-1395

NEW YORK

Arrow Electronics, Inc.
25 Hub Drive
Melville 11747
Tel: (516) 694-6800
TWX: 510-224-6126
FAX: 516-391-1401

†Arrow Electronics, Inc.
3375 Brighton-Henrietta Townline Rd.
Rochester 14623
Tel: (716) 427-0300
FAX: 716-427-0735

Arrow Electronics, Inc.
20 Oser Avenue
Hauppauge 11788
Tel: (516) 231-1000
FAX: 516-231-1072

Hamilton/Avnet Electronics
2060 Townline Rd.
Rochester 14623
Tel: (716) 475-9130
FAX: 716-475-9119

†Hamilton/Avnet Electronics
103 Twin Oaks Drive
Syracuse 13206
Tel: (315) 437-2641
FAX: 315-432-0740

†Hamilton/Avnet Electronics
933 Motor Parkway
Hauppauge 11788
Tel: (516) 231-9800
FAX: 516-434-7426

†MTI Systems Sales
38 Harbor Park Drive
P.O. Box 271
Port Washington 11050
Tel: (516) 621-6200
FAX: 516-625-3039

†Pioneer Northeast Electronics
68 Corporate Dr.
Binghamton 13904
Tel: (607) 722-9300
FAX: 607-722-9562

†Pioneer Northeast Electronics
60 Crossway Park West
Woodbury, Long Island 11797
Tel: (516) 921-8700
TWX: 510-221-2184
FAX: 516-921-2143

†Pioneer Northeast Electronics
840 Fairport Park
Fairport 14450
Tel: (716) 381-7070
FAX: 716-381-5955

NORTH CAROLINA

†Arrow Electronics, Inc.
5240 Greens Dairy Road
Raleigh 27604
Tel: (919) 876-3132
FAX: 919-876-3132, ext. 200

†Hamilton/Avnet Electronics
3510 Spring Forest Drive
Raleigh 27609
Tel: (919) 878-0819
TWX: 510-928-1836

NORTH CAROLINA (Cont'd.)

Pioneer Electronics
9801 A-Southern Pine Blvd.
Charlotte 28217
Tel: (704) 527-8188
TWX: 810-621-0366

OHIO

Arrow Electronics, Inc.
7620 McEwen Road
Centerville 45459
Tel: (513) 435-5563
FAX: 513-435-2049

†Arrow Electronics, Inc.
6238 Cochran Road
Solon 44139
Tel: (216) 248-3990
FAX: 216-248-1106

Hamilton/Avnet Electronics
777 Brooksedge Blvd.
Westerville 43081
Tel: (614) 882-7004
FAX: 614-882-8650

†Hamilton/Avnet Electronics
954 Senate Drive
Dayton 45459
Tel: (513) 439-6700
FAX: 513-439-6711

†Hamilton/Avnet Electronics
30325 Bainbridge Rd., Bldg. A
Solon 44139
Tel: (216) 349-5100
FAX: 216-349-1894

†Pioneer Electronics
4433 Interpoint Blvd.
Dayton 45424
Tel: (513) 236-9900
FAX: 513-236-8133

†Pioneer Electronics
4800 E. 131st Street
Cleveland 44105
Tel: (216) 587-3600
TWX: 810-422-2211
FAX: 216-587-3906

OKLAHOMA

Arrow Electronics, Inc.
3158 S. 108 East Ave., Ste. 210
Tulsa 74146
Tel: (918) 665-7700
FAX: 918-665-7700

OREGON

†Almac Electronics Corp.
1885 N.W. 169th Place
Beaverton 97006
Tel: (503) 629-8090
FAX: 503-645-0611

†Hamilton/Avnet Electronics
6024 S.W. Jean Road
Bldg. C, Suite 10
Lake Oswego 97034
Tel: (503) 635-7848
FAX: 503-636-1327

Wyle Distribution Group
5250 N.E. Elam Young Parkway
Suite 600
Hillsboro 97124
Tel: (503) 640-6000
FAX: 503-640-5846

PENNSYLVANIA

Arrow Electronics, Inc.
650 Seco Road
Monroeville 15146
Tel: (412) 856-7000
FAX: 412-856-5777

Hamilton/Avnet Electronics
2800 Liberty Ave., Bldg. E
Pittsburgh 15222
Tel: (412) 281-4150
FAX: 412-281-8662

PENNSYLVANIA (Cont'd.)

Pioneer Electronics
259 Kappa Drive
Pittsburgh 15238
Tel: (412) 782-2300
TWX: 710-795-3122
FAX: 412-963-8255

†Pioneer Electronics
261 Gibralter Road
Horsham 19044
Tel: (215) 674-4000
TWX: 510-665-6778
FAX: 215-674-3107

TEXAS

†Arrow Electronics, Inc.
3220 Commander Drive
Carrollton 75006
Tel: (214) 380-6464
FAX: 214-248-7208

†Arrow Electronics, Inc.
10899 Kinghurst Dr.
Suite 100
Houston 77099
Tel: (713) 530-4700
FAX: 713-568-8518

†Arrow Electronics, Inc.
2227 W. Braker Lane
Austin 78758
Tel: (512) 835-4180
FAX: 512-832-9875

†Hamilton/Avnet Electronics
1807A W. Braker Lane
Austin 78758
Tel: (512) 837-8911
FAX: 512-339-6232

†Hamilton/Avnet Electronics
2111 W. Walnut Hill Lane
Irving 75038
Tel: (214) 550-6111
FAX: 214-550-6172

†Hamilton/Avnet Electronics
4850 Wright Road, Ste. 190
Stafford 77477
Tel: (713) 240-7733
FAX: 713-240-0582

Kierulff Electronics, Inc.
2010 Merritt Dr.
Garland 75040
Tel: (214) 840-0110
FAX: 214-278-0928

†Pioneer Electronics
1826-D Kramer Lane
Austin 78758
Tel: (512) 835-4000
FAX: 512-835-9829

†Pioneer Electronics
13710 Omega Road
Dallas 75244
Tel: (214) 386-7300
FAX: 214-490-6419

†Pioneer Electronics
5853 West Drive
Houston 77036
Tel: (713) 988-5555
FAX: 713-988-1732

UTAH

†Hamilton/Avnet Electronics
1585 West 2100 South
Salt Lake City 84119
Tel: (801) 972-2800
FAX: 801-974-9675

Kierulff Electronics, Inc.
1946 W. Parkway Blvd.
Salt Lake City 84119
Tel: (801) 973-6913
FAX: 801-972-0200

Wyle Distribution Group
1325 West 2200 South
Suite E
Salt Lake City 84119
Tel: (801) 974-9953
FAX: 801-972-2524

WASHINGTON

†Almac Electronics Corp.
14360 S.E. Eastgate Way
Bellevue 98007
Tel: (206) 643-9992
FAX: 206-643-9709

Arrow Electronics, Inc.
14320 N.E. 21st Street
Bellevue 98007
Tel: (206) 643-4800
FAX: 206-746-3740

Hamilton/Avnet Electronics
14212 N.E. 21st Street
Bellevue 98005
Tel: (206) 453-5874
FAX: 206-643-0086

Wyle Distribution Group
1750 132nd Ave., N.E.
Bellevue 98005
Tel: (206) 453-8300
FAX: 206-453-4071

WISCONSIN

†Arrow Electronics, Inc.
200 N. Patrick Blvd., Ste. 100
Brookfield 53005
Tel: (414) 792-0150
FAX: 414-792-0156

Hamilton/Avnet Electronics
2975 Moorland Road
New Berlin 53151
Tel: (414) 784-4510
FAX: 414-784-9509

Kierulff Electronics, Inc.
2238-E W. Bluemound Rd.
Waukesha 53186
Tel: (414) 784-8160
FAX: 414-784-0409

CANADA

ALBERTA

Hamilton/Avnet Electronics
2816 21st Street N.E.
Calgary T2E 6Z2
Tel: (403) 250-9380
FAX: 403-250-1591

Zentronics
6815 8th Street, N.E., Ste. 100
Calgary T2E 7H7
Tel: (403) 295-8838
FAX: 403-295-8714

BRITISH COLUMBIA

Hamilton/Avnet Electronics
2550 Boundary Rd., Ste. 115
Burnaby V5M 3Z3
Tel: (604) 437-6667
FAX: 604-437-4712

Zentronics
108-11400 Bridgeport Road
Richmond V6X 1T2
Tel: (604) 273-5575
FAX: 604-273-2413

MANITOBA

Zentronics
60-1313 Border Street
Winnipeg R3H 0X4
Tel: (204) 694-1957
FAX: 204-633-9255

ONTARIO

Arrow Electronics Inc.
1093 Meyerside Dr.
Unit 2
Mississauga LST 1M4
Tel: (416) 672-7769
FAX: 416-672-0489

Arrow Electronics Inc.
Nepean K2E 7W5
Tel: (613) 226-6903
FAX: 613-723-2018

†Hamilton/Avnet Electronics
6845 Rexwood Road
Units 3-5
Mississauga L4V 1R2
Tel: (416) 677-7432
FAX: 416-677-0940

Hamilton/Avnet Electronics
3688 Nashua Dr.
Units 9 and 10
Mississauga L4V 1M5
Tel: (416) 677-0484
FAX: 416-677-0627

†Hamilton/Avnet Electronics
190 Colonnade Road South
Nepean K2E 7J5
Tel: (613) 226-1700
FAX: 613-226-1184

†Zentronics
8Tilbury Court
Brampton L6T 3T4
Tel: (416) 451-9600
FAX: 416-451-8320

†Zentronics
155 Colonnade Road
Unit 17
Nepean K2E 7K1
Tel: (613) 226-8840
FAX: 613-226-8350

SASKATCHEWAN

Zentronics
173-1222 Alberta Avenue
Saskatoon S7K 1R4
Tel: (306) 955-2202, 2207
FAX: (306) 244-3731

QUEBEC

†Arrow Electronics Inc.
4050 Jean Talon Quest
Montreal H4P 1W1
Tel: (514) 735-5511
FAX: 514-341-4821

Arrow Electronics Inc.
909 Charest Blvd.
Quebec 61N 269
Tel: (418) 687-4231
FAX: 418-687-5348

Hamilton/Avnet Electronics
2795 Rue Halpern
St. Laurent H4S 1P8
Tel: (514) 335-1000
FAX: 514-335-2481

Zentronics
817 McCafrey St.
St. Laurent H4T 1N4
Tel: (514) 737-9700
FAX: 514-737-5212

†Microcomputer System Technical Distributor Centers

EUROPEAN SALES OFFICES

DENMARK

Intel
Glentevej 61, 3rd Floor
2400 Copenhagen NV
Tel: (01) 19 80 33
TLX: 19567

FINLAND

Intel
Ruosilantie 2
00390 Helsinki 39
Tel: (0) 54 46 44
TLX: 123332

FRANCE

Intel
1, rue Edison-BP 303
78054 St Quentin-en-Yvelines Cedex
Tel: (1) 30 57 70 00
TLX: 699016

Intel
Immeuble BBC
4, Quai des Etroits
69005 Lyon
Tel: 78 42 40 89
TLX: 305153

WEST GERMANY

Intel*
Seidlstrasse 27
8000 Muenchen 2
Tel: (089) 5 38 90
TLX: 523177

Intel
Hohenzollern Strasse 5
3000 Hannover 1
Tel: (051) 1 34 40 81
TLX: 923625

Intel
Abraham Lincoln Strasse 16-18
6200 Wiesbaden
Tel: (061) 217 60 50
TLX: 4186183

Intel
Bruckstrasse 61
7012 Feilbach
Stuttgart
Tel: (071) 158 00 82
TLX: 7254826

ISRAEL

Intel*
Attidim Industrial Park
P.O. Box 43202
Tel Aviv 61430
Tel: (03) 49 80 80
TLX: 371215

ITALY

Intel*
Milanofiori Palazzo E
20090 Assago
Milano
Tel: (02) 824 40 71
TLX: 341286

NETHERLANDS

Intel*
Alexander Poort Building
Marten Meesweg 93
3068 AV Rotterdam
Tel: (010) 421 23 77
TLX: 22283

NORWAY

Intel
Hvamveien 4-P.O. Box 92
2013 Skjetten
Tel: (6) 842 420
TLX: 78018

SPAIN

Intel
Calle Zurbaran no. 28-1 Isq
28010 Madrid
Tel: (1) 410 40 04
TLX: 46880

SWEDEN

Intel*
Dalvagen 24
17136 Solna
Tel: (08) 734 01 00
TLX: 12261

SWITZERLAND

Intel*
Talackerstrasse 17
8065 Zurich
Tel: (01) 829 29 77
TLX: 57989

UNITED KINGDOM

Intel*
Pipers Way
Swindon, Wiltshire SN3 1RJ
Tel: (0793) 69 60 00
TLX: 444447

EUROPEAN DISTRIBUTORS/REPRESENTATIVES

AUSTRIA

Bacher Electronics GmbH
Rotenmuehlgasse 26
1120 Wien
Tel: (0222) 835 64 60
TLX: 131532

BELGIUM

Inelco Belgium S.A.
Av. des Croix de Guerre 94
1120 Bruxelles
Tel: (02) 216 01 60
TLX: 64475

DENMARK

ITT-Multikomponent A/S
Naverland 29
2600 Glostrup
Tel: (02) 45 66 45
TLX: 33355

FINLAND

OY Fintronic AB
Melkonkatu 24A
00210 Helsinki 21
Tel: (0) 692 60 22
TLX: 124224

FRANCE

Generim
Z.A. de Courtaboeuf
Av. de la Baltique-BP 88
91943 Les Ulis Cedex
Tel: (1) 69 07 78 78
TLX: 691700

Jermyn S.A.
73-79, rue des Solets
Silic 585
94663 Rungis Cedex
Tel: (1) 45 60 04 00
TLX: 260967

Metrologie
Tour d'Asnieres
4, av. Laurent-Cely
92606 Asnieres Cedex
Tel: (1) 47 90 62 40
TLX: 611448

Tekelec-Airtronic
Cite des Bruyeres
Rue Carle-Vernet - BP 2
92310 Sevres
Tel: (1) 45 34 75 35
TLX: 204552

WEST GERMANY

Electronic 2000 Vertriebs-AG
Stahlgruberring 12
8000 Muenchen 82
Tel: (089) 42 00 10
TLX: 522561

ITT Multikomponent GmbH
Bahnhofstrasse 44
7141 Moeglingen
Tel: (071) 41 48 79
TLX: 7264399

Jermyn GmbH
Im Dachsstueck 9
6250 Limburg
Tel: (064) 31 50 80
TLX: 415257-0

Metrologie GmbH
Meglingerstrasse 49
8000 Muenchen 71
Tel: (089) 78 04 20
TLX: 5213189

Proelectron Vertriebs GmbH
Max Planck Strasse 1-3
6072 Dreieich
Tel: (061) 03 30 43 43
TLX: 417972

IRELAND

Micro Marketing
Glenageary Office Park
Glenageary
Co Dublin
Tel: (1) 85 62 88
TLX: 31584

ISRAEL

Eastronics Ltd.
11 Rozanis Street
P.O. Box 39300
Tel Aviv 61392
Tel: (03) 47 51 51
TLX: 33638

ITALY

Intesi Corporation Italia S.A.
Milanofiori Palazzo E/5
20090 Assago
Milano
Tel: (02) 82 47 01
TLX: 311351

ITALY (Cont'd.)

Lasi Elettronica S.p.a.
Viale Fulvio Testi 126
20092 Cinisello Balsamo
Milano
Tel: (02) 244 00 12
TLX: 352040

NETHERLANDS

Koning en Hartman
Energieweg 1
2627 AP Delft
Tel: (015) 60 99 06
TLX: 38250

NORWAY

Nordisk Elektronik A/S
P.O. Box 122
Smedsvingen 4
1364 Hvalstad
Tel: (2) 84 62 10
TLX: 77546

PORTUGAL

Ditram
Av. M. Bombarda, 133-1 D
1000 Lisboa
Tel: (1) 54 53 13
TLX: 14182

SPAIN

ATD Electronica S.A.
Plaza Ciudad de Viena no. 6
28040 Madrid
Tel: (1) 234 40 00
TLX: 42477

ITT-SESA
Calle Miguel Angel no. 21-3
28010 Madrid
Tel: (1) 419 09 57
TLX: 27461

SWEDEN

Nordisk Elektronik A.B.
Huvudstagatan 1
P.O. Box 1409
17127 Solna
Tel: (8) 734 97 70
TLX: 10547

SWITZERLAND

Industrade A.G.
Hertistrasse 31
8304 Wallisellen
Tel: (01) 8 30 50 40
TLX: 56788

UNITED KINGDOM

Accent Electronic Components Ltd.
Jubilee House, Jubilee Way
Letchworth, Herts SG6 1QH
Tel: (0462) 68 66 66
TLX: 626923

Bytech Comway Ltd.
Unit 2 The Western Centre
Western Road
Bracknell
Berks RG12 1RW
Tel: (0344) 48 22 11
TLX: 848215

Jermyn
Vestry Estate
Otford Road
Sevenoaks
Kent TN14 5EU
Tel: (0732) 45 01 44
TLX: 95142

Rapid Silicon
Rapid House
Denmark St.
High Wycombe
Bucks HP11 2ER
Tel: (0494) 44 22 66
TLX: 837931

Rapid Systems
Rapid House
Denmark St.
High Wycombe
Bucks HP11 2ER
Tel: (0494) 45 02 44
TLX: 837931

YUGOSLAVIA

H.R. Microelectronics Corp.
2005 de la Cruz Blvd., Ste. 223
Santa Clara, CA 95050
U.S.A.
Tel: (408) 988-0286
TLX: 387452

*Field Application Location

INTERNATIONAL SALES OFFICES

AUSTRALIA

Intel Australia Pty. Ltd.*
Spectrum Building
200 Pacific Hwy., Level 6
Crows Nest, NSW, 2065
Tel: (2) 957-2744
TLX: 20097
FAX: (2) 923-2632

BRAZIL

Intel Semicondutores do Brasil LTDA
Av. Paulista, 1159-CJS 404/405
01311 - Sao Paulo - S.P.
Tel: 55-11-287-5899
TLX: 1153146
FAX: 55-11-212-7631

CHINA

Intel PRC Corporation
15/F, Office 1, Citic Bldg.
Jian Guo Men Wai Street
Beijing, PRC
Tel: (1) 500-4850
TLX: 22947 INTEL CN
FAX: (1) 500-2953

HONG KONG

Intel Semiconductor Ltd.*
1701-3 Connaught Centre
1 Connaught Road
Tel: (5) 844-4555
TWX: 63869 ISLHK HX
FAX: (5) 294-589

JAPAN

Intel Japan K.K.
5-6 Tokodai Toyosato-machi
Tsukuba-gun, Ibaraki-ken 300-26
Tel: 029747-8511
TLX: 3656-160
FAX: 029747-8450

Intel Japan K.K.*
Daiichi Mitsugi Bldg.
1-8889 Fuchu-cho
Fuchu-shi, Tokyo 183
Tel: 0423-60-7871
FAX: 0423-60-0315

Intel Japan K.K.*
Flower-Hill Shin-machi Bldg.
1-23-9 Shinmachi
Setagaya-ku, Tokyo 154
Tel: 03-426-2231
FAX: 03-427-7620

Intel Japan K.K.*
Bldg. Kumagaya
2-69 Hon-cho
Kumagaya-shi, Saitama 360
Tel: 0485-24-6871
FAX: 0485-24-7518

Intel Japan K.K.*
Mitsui-Seimei Musashi-kosugi Bldg.
915 Shinmaruko, Nakahara-ku
Kawasaki-shi, Kanagawa 211
Tel: 044-733-7011
FAX: 044-733-7010

JAPAN (Cont'd)

Intel Japan K.K.
Nihon Seimei Atsugi Bldg.
1-2-1 Asahi-machi
Atsugi-shi, Kanagawa 243
Tel: 0462-29-3731
FAX: 0462-29-3781

Intel Japan K.K.*
Ryokuchi-Eki Bldg.
2-4-1 Terauchi
Toyonaka-shi, Osaka 560
Tel: (06) 863-1091
FAX: 06-863-1084

Intel Japan K.K.
Shinmaru Bldg.
1-5-1 Marunouchi
Chiyoda-ku, Tokyo 100
Tel: 03-201-3621
FAX: 03-201-6850

Intel Japan K.K.
Tokai Bldg.
1-16-30 Meieki Minami
Nakamura-ku, Nagoya-shi
Aichi 450
Tel: 052-561-5181
FAX: 052-561-5317

KOREA

Intel Technology Asia Ltd.
Room 906, Singsong Bldg.
25-4, Yoido-Dong, Youngdeungpo-ku
Seoul 150
Tel: (2) 784-8186
TLX: 29312 INTELKO
FAX: (2) 784-8096

SINGAPORE

Intel Singapore Technology, Ltd.
101 Thomson Road #21-06
Goldhill Square
Singapore 1130
Tel: 250-7811
TLX: 39921 INTEL
FAX: 250-9256

TAIWAN

Intel Technology (Far East) Ltd.
Taiwan Branch
10/F., No. 205, Tun Hua N. Road
Taipei, R.O.C.
Tel: 886-2-716-9660
TLX: 13159 INTELTWN
FAX: 886-2-717-2455

INTERNATIONAL DISTRIBUTORS/REPRESENTATIVES

ARGENTINA

DAFSYS S.R.L.
Chacabuco, 90-4 PISO
1069-Buenos Aires
Tel: 54-1-334-1871
54-1-34-7726
TLX: 25472

Reycom Electronica S.R.L.
Arcos 3631
1429-Buenos Aires
Tel: 54 (1) 701-4462/66
FAX: 54 (1) 11-1722
TLX: 22122

AUSTRALIA

Total Electronics
P.M.B. 250
9 Harker Street
Burwood, Victoria 3125
Tel: 61-3-288-4044
TLX: AA 31261

Total Electronics
P.O. Box 139
Artamon, N.S.W. 2064
Tel: 61-02-438-1855
TLX: 26297

BRAZIL

Elebra Microelectronica
R. Geraldo Flausino Gomes, 78
9 Andar
04575 - Sao Paulo - S.P.
Tel: 55-11-534-9522
TLX: 1154591 or 1154593BR
FAX: 55-11-534-9637

CHILE

DIN Instruments
Suecia 2323
Casilla 6055, Correo 22
Santiago
Tel: 56-2-225-8139
TLX: 440422 RUDY CZ

CHINA

Novel Precision Machinery Co., Ltd.
Flat D, 20 Kingsford Ind. Bldg.
Phase 1, 26 Kwai Hei Street
N.T., Kowloon
Hong Kong
Tel: 852-0-223-222
TWX: 39114 JINMI HX
FAX: 852-0-261-602

CHINA (Cont'd)

Schmidt & Co. Ltd.
18/F Great Eagle Centre
23 Harbour Road
Wanchai, Hong Kong
Tel: 852-5-833-0222
TWX: 74766 SCHMC HX
FAX: 852-5-891-8754

INDIA

Micronic Devices
Arun Complex
No. 65 D.V.G. Road
Basavanagudi
Bangalore 560 004
Tel: 91-812-600-631
TLX: 0845-8332 MD BG IN

Micronic Devices
403, Gagan Deep
12, Rajendra Place
New Delhi 110 008
Tel: 91-58-97-71
TLX: 03163235 MDND IN

Micronic Devices
No. 516 5th Floor
Swastik Chambers
Sion, Chambray Road
Bombay 400 071
Tel: 91-52-39-63
TLX: 9531 171447 MDEV IN

JAPAN

Asahi Electronics Co. Ltd.
KMM Bldg. 2-14-1 Asano
Kokurakita-ku
Kitakyushu-shi 802
Tel: 093-511-6471
FAX: 093-551-7861

C. Itoh Techno-Science Co., Ltd.
C. Itoh Bldg., 2-5-1 Kita-Aoyama
Minato-ku, Tokyo 107
Tel: 03-497-4840
FAX: 03-497-4969

JAPAN (Cont'd)

Dia Semicon Systems, Inc.
Wacore 64, 1-37-8 Sangenjaya
Setagaya-ku, Tokyo 154
Tel: 03-487-0386
FAX: 03-487-8088

Okaya Koki
2-4-18 Sakae
Naka-ku, Nagoya-shi 460
Tel: 052-204-2911
FAX: 052-204-2901

Ryoyo Electro Corp.
Konwa Bldg.
1-12-22 Tsukiji
Chuo-ku, Tokyo 104
Tel: 03-546-5011
FAX: 03-546-5044

KOREA

J-Tek Corporation
6th Floor, Government Pension Bldg.
24-3 Yoido-Dong
Youngdeungpo-ku
Seoul 150
Tel: 82-2-782-8039
TLX: 25299 KODIGIT
FAX: 82-2-784-8391

Samsung Semiconductor &
Telecommunications Co., Ltd.
150, 2-KA, Tafpyung-ro, Chung-ku
Seoul 100
Tel: 82-2-751-3987
TLX: 27970 KORSST
FAX: 82-2-753-0967

MEXICO

Dicopel S.A.
Tochtli 368 Fracc. Ind. San Antonio
Azcapotzalco
C.P. 02760-Mexico, D.F.
Tel: 52-5-561-3211
TLX: 1773790 DICOME

NEW ZEALAND

Northrup Instruments & Systems Ltd.
459 Kyber Pass Road
P.O. Box 9464, Newmarket
Auckland 1
Tel: 64-9-501-219, 501-801
TLX: 21570 THERMAL

Northrup Instruments & Systems Ltd.
P.O. Box 2406
Wellington 856658
Tel: 64-4-856-658
TLX: NZ3380
FAX: 64-4-857276

SINGAPORE

Francotone Electronics Pte Ltd.
17 Harvey Road #04-01
Singapore 1336
Tel: 283-0888, 289-1618
TWX: 56541 FRELS
FAX: 2895327

SOUTH AFRICA

Electronic Building Elements, Pty. Ltd.
P.O. Box 4609
Pine Square, 18th Street
Hazelwood, Pretoria 0001
Tel: 27-12-469921
TLX: 3-227786 SA

TAIWAN

Mitac Corporation
No. 585, Ming Shen East Rd.
Taipei, R.O.C.
Tel: 886-2-501-8231
FAX: 886-2-501-4265

VENEZUELA

P. Benavides S.A.
Avilanes a Rio
Residencia Kamarata
Locales 4 AL 7
La Candelaria, Caracas
Tel: 58-2-571-0396
TLX: 28450
FAX: 58-2-572-3321

*Field Application Location

DOMESTIC SERVICE OFFICES

ALABAMA

Intel Corp.
5015 Bradford Drive, #2
Huntsville 35805
Tel: (205) 830-4010

ARIZONA

Intel Corp.
11225 N. 28th Dr., #D214
Phoenix 85029
Tel: (602) 869-4980

Intel Corp.
500 E. Fry Blvd., Suite M-15
Sierra Vista 85635
Tel: (602) 459-5010

ARKANSAS

Intel Corp.
P.O. Box 206
Ulm 72170
Tel: (501) 241-3264

CALIFORNIA

Intel Corp.
21515 Vanowen St.
Suite 116
Canoga Park 91303
Tel: (818) 704-8500

Intel Corp.
2250 E. Imperial Highway
Suite 218
El Segundo 90245
Tel: 1-800-468-3548

Intel Corp.
1900 Prairie City Rd.
Folsom 95630-9597
Tel: (916) 351-6143

Intel Corp.
2000 E. 4th Street
Suite 110
Santa Ana 92705
Tel: (714) 835-5789
TWX: 910-595-2475

Intel Corp.
2700 San Tomas Expressway
Santa Clara 95051
Tel: (408) 970-1740

Intel Corp.
4350 Executive Drive
Suite 105
San Diego 92121
Tel: (619) 452-5880

COLORADO

Intel Corp.
650 South Cherry
Suite 915
Denver 80222
Tel: (303) 321-8086
TWX: 910-931-2289

CONNECTICUT

Intel Corp.
26 Mill Plain Road
Danbury 06811
Tel: (203) 748-3130
TWX: 710-456-1199

FLORIDA

Intel Corp.
1500 N.W. 62, Suite 104
Ft. Lauderdale 33309
Tel: (305) 771-0600
TWX: 510-956-9407

Intel Corp.
242 N. Westmonte Drive
Suite 105
Altamonte Springs 32714
Tel: (305) 869-5588

GEORGIA

Intel Corp.
3280 Pointe Parkway
Suite 200
Norcross 30092
Tel: (404) 441-1171

ILLINOIS

Intel Corp.
300 N. Martingale Rd.
Suite 300
Schaumburg 60194
Tel: (312) 310-5733

INDIANA

Intel Corp.
8777 Purdue Rd., #125
Indianapolis 46268
Tel: (317) 875-0623

KANSAS

Intel Corp.
8400 W. 110th Street
Suite 170
Overland Park 66210
Tel: (913) 345-2727

KENTUCKY

Intel Corp.
3525 Tatescreek Road, #51
Lexington 40502
Tel: (606) 272-6745

MARYLAND

Intel Corp.
5th Floor
7833 Walker Drive
Greenbelt 20770
Tel: (301) 441-1020

MASSACHUSETTS

Intel Corp.
Westford Corp. Center
3 Carlisle Road
Westford 01886
Tel: (617) 692-1060

MICHIGAN

Intel Corp.
7071 Orchard Lake Road
Suite 100
West Bloomfield 48033
Tel: (313) 851-8905

MISSOURI

Intel Corp.
4203 Earth City Expressway
Suite 143
Earth City 63045
Tel: (314) 291-2015

NEW JERSEY

Intel Corp.
385 Sylvan Avenue
Englewood Cliffs 07632
Tel: (201) 567-0821
TWX: 710-991-8593

Intel Corp.
Raritan Plaza III
Raritan Center
Edison 08817
Tel: (201) 225-3000

NORTH CAROLINA

Intel Corp.
2306 W. Meadowview Road
Suite 206
Greensboro 27407
Tel: (919) 294-1541

Intel Corp.
2700 Wycliff Rd., Suite 102
Raleigh 27607
Tel: (919) 781-8022

OHIO

Intel Corp.
Chagrin-Brainard Bldg.
Suite 305
28001 Chagrin Boulevard
Cleveland 44122
Tel: (216) 464-6915
TWX: 810-427-9298

Intel Corp.
6500 Poe
Dayton 45414
Tel: (513) 890-5350

OREGON

Intel Corp.
15254 N.W. Greenbrier Parkway, Bldg. B
Beaverton 01886
Tel: (503) 645-8051
TWX: 910-467-8741

Intel Corp.
5200 N.E. Elam Young Parkway
Hillsboro 97123
Tel: (503) 681-8080

PENNSYLVANIA

Intel Corp.
201 Penn Center Boulevard
Suite 301 W
Pittsburgh 15235
Tel: (313) 354-1540

TEXAS

Intel Corp.
313 E. Anderson Lane
Suite 314
Austin 78752
Tel: (512) 454-3628
TWX: 910-874-1347

Intel Corp.
12300 Ford Road
Suite 380
Dallas 75234
Tel: (214) 241-2620
TWX: 910-860-5617

Intel Corp.
8815 Dyer St., Suite 225
El Paso 79904
Tel: (915) 751-0186

VIRGINIA

Intel Corp.
1603 Santa Rosa Rd., #109
Richmond 23288
Tel: (804) 282-5668

WASHINGTON

Intel Corp.
110 110th Avenue N.E.
Suite 510
Bellevue 98004
Tel: 1-800-468-3548
TWX: 910-443-3002

WISCONSIN

Intel Corp.
330 S. Executive Dr.
Suite 102
Brookfield 53005
Tel: (414) 784-8087

CANADA

Intel Corp.
190 Attwell Drive, Suite 500
Rexdale, Ontario
Canada M9W 6H8
Tel: (416) 675-2105

Intel Corp.
620 St. Jean Blvd.
Pointe Claire, Quebec
Canada H9R 3K3
Tel: (514) 694-9130

Intel Corp.
2650 Queensview Drive, #250
Ottawa, Ontario,
Canada K2B 8H6
Tel: (613) 829-9714

CUSTOMER TRAINING CENTERS

CALIFORNIA

2700 San Tomas Expressway
Santa Clara 95051
Tel: (408) 970-1700

ILLINOIS

300 N. Martingale, #300
Schaumburg 60173
Tel: (312) 310-5700

MASSACHUSETTS

3 Carlisle Road
Westford 01886
Tel: (617) 692-1000

MARYLAND

7833 Walker Dr., 4th Floor
Greenbelt 20770
Tel: (301) 220-3380

SYSTEMS ENGINEERING OFFICES

CALIFORNIA

2700 San Tomas Expressway
Santa Clara 95051
Tel: (408) 986-8086

ILLINOIS

300 N. Martingale, #300
Schaumburg 60173
Tel: (312) 310-8031

MASSACHUSETTS

3 Carlisle Road
Westford 01886
Tel: (617) 692-3222

NEW YORK

300 Motor Parkway
Hauppauge 11788
Tel: (516) 231-3300

UNITED STATES
Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95051

JAPAN
Intel Japan K.K.
5-6 Tokodai Toyosato-machi
Tsukuba-gun, Ibaraki-ken 300-26

FRANCE
Intel Paris
1 Rue Edison, BP 303
78054 Saint-Quentin-en-Yvelines Cedex

UNITED KINGDOM
Intel Corporation (U.K.) Ltd.
Pipers Way
Swindon
Wiltshire, England SN3 1RJ

WEST GERMANY
Intel Semiconductor GmbH
Seidlstrasse 27
D-8000 Muenchen 2

HONG KONG
Intel Semiconductor Ltd.
1701-3 Connaught Centre
1 Connaught Road

CANADA
Intel Semiconductor of Canada, Ltd.
190 Attwell Drive, Suite 500
Rexdale, Ontario M9W 6H8

Printed in U.S.A./0188/40K/RRD/LK
ASIC

ISBN 1-55512-028-8